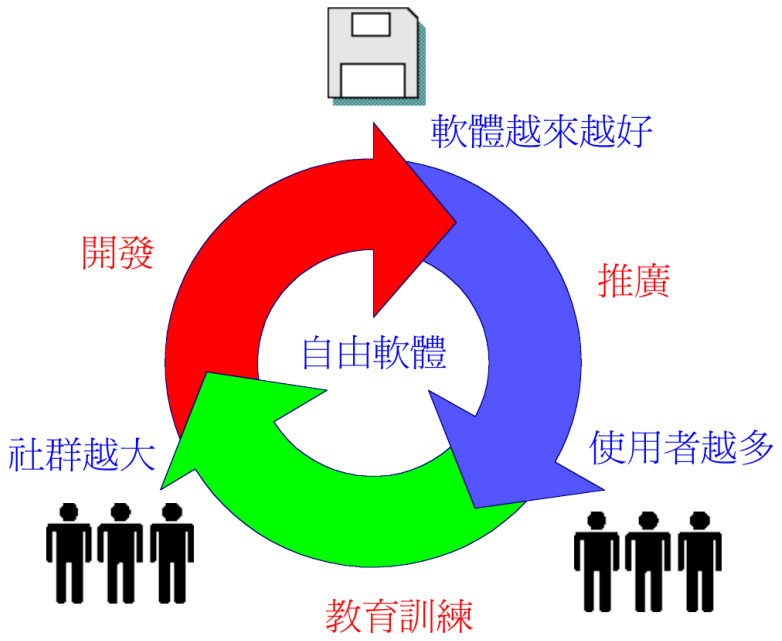
Linux & 自由軟體總藍圖



軟體自由協會

Version 1.01

自由軟體總藍圖

中華民國軟體自由協會著

編輯 ：葉平、廖志宇

作者 ： 謝東翰、吳鴻煦、鄭大慶、李柏鋒、李健秋、陳開基、胡師賢、李春和、葉平

版本歷史 ：

2001 年 8 月　　第一版

本自由軟體總藍圖是以自由軟體 LaTeX 在 Linux 平台上編輯排版完成。

前言

以 Linux 爲首的自由軟體熱潮正席捲著全球，並不斷擴大對國際知識經濟體系與世界各國資訊工業的影響。海峽對岸更將 Linux 與自由軟體視爲在世界各國知識經濟的競賽中，縮小數位落差、提升軟體實力和增強國家資訊安全的強力武器；身爲資訊大國的台灣當然更不能缺席於這股知識經濟的浪潮。

民國九十年五月十七日，由行政院國家資訊通信發展推動小組召開「中文版 Linux 未來的發展工作」會議，邀集台北市電腦公會、資策會、軟體自由協會．．．等單位，共同討論台灣 Linux 產業的發展方向。會中決議應先有一份藍圖，以作爲往後之依據。軟體自由協會有幸得以擔任此藍圖規劃之角色，並號召社群參與，因而產生了這份『自由軟體總藍圖』。

本藍圖分三大部分：第一部分「自由軟體簡介」說明自由軟體的本質、優勢、運作方式和未來趨勢；第二部分「自由軟體發展藍圖」以「願景」、「目標」及「策略」爲架構，積極描繪二十一世紀自由軟體在台灣的面貌，希望對施政當局在作相關決策時有所助益；第三部分「自由軟體元件細圖」對各種自由軟體分門別類做簡介，可作爲技術人員的參考，我們寄望此部分成爲一社群計劃，持續更新加深。

藉著本『自由軟體總藍圖』的規劃，我們希望執政當局正視自由軟體所帶來的資訊安全、技術紮根和控制權在握等效益，著力扶持，使全台灣資訊從業人員和電腦玩家都成爲自由軟體社群的一份子，增進軟體品質，帶動開放源碼產業起飛，成就「自由軟體新故鄉」的願景。

誌謝

這份自由軟體總藍圖，是靠著許多人的貢獻才完成的，除了感謝作者和編輯苦心生出的文字之外，一定要感謝台大資工系的許清琦教授提供大家一個固定會面的場所，並不時打氣；當然第一次開會好吃的便當是無法忘懷的。我們也要謝謝黃志偉、劉政和胡崇偉提供的大綱，對於釐清章節發揮了很大的作用。黃志偉的 LaTeX 範例對我們排版工作有很大的助益。還要謝謝鄭原眞提供的 mailing list 服務，讓我們可以有效地溝通。不可以忘記的，是讓這一切事情成爲可能的自由軟體社群，尤其是 CLE，本藍圖就是在 CLE 0.9p1 的平台上完成的。最後要謝謝政府，給軟體自由協會這個機會來以社群和產業界的角度規劃這份總藍圖。

目 錄

第壹篇　自由軟體簡介 **1**

第一章　自由軟體簡介 **3**

1.1　什麼是自由軟體？ 4

1.2　自由軟體的優勢 5

1.3　自由軟體的運作方式 6

1.4　未來的趨勢 8

第貳篇　自由軟體發展藍圖 **9**

第二章　自由軟體發展藍圖 **11**

2.1　支持教育推廣 12

2.2　進行基礎建設 15

2.3　發展重點領域 17

第參篇　自由軟體元件細圖 **21**

第三章　自由軟體與微軟視窗平台的對照表 **23**

3.1　簡介 23

3.2　解決方案之比較 25

3.3　開發系統的比較 27

3.4 嵌入式系統 32
3.5 自由軟體和其他之實測比較 33

第四章 網路應用 (Network Applications) 37
4.1 網路應用伺服系統 37
4.2 網路共用程式庫 38
4.3 網際網路伺服器 39
4.4 網路檔案系統 40
4.5 網路管理 41
4.6 網路安全工具 42
4.7 防火牆 42

第五章 資料管理 (Data Management) 43
5.1 資料庫 (Database) 43
5.1.1 MySQL 資料庫 43
5.1.2 PostgreSQL 52
5.1.3 其他免費資料庫 60
5.2 檔案系統 (File System) 61
5.3 編碼 (Encoding) 75
5.3.1 電腦編碼系統簡介 75
5.3.2 GNU/Linux 下的編碼與字集 78
5.3.3 GNU/Linux 的轉碼系統 79

第六章 開發環境 83
6.1 總論 83
6.2 編譯器家族 GCC 84
6.3 各種程式工具 binutils 85
6.4 除錯工具 gdb 88

6.5 編譯自動化 make 89
6.6 跨平台輔助工具 91
6.7 程式維護 cvs 96
6.8 各式各樣的直譯式語言 98
6.9 程式文件維護 103
6.10 整合開發環境與其他開發環境 104

第七章 程式庫 (Library) 109

7.1 總論 109
7.2 系統呼叫 110
7.3 標準函式庫 110
7.3.1 簡介 110
7.3.2 GLIBC 的規格 112
7.3.3 GLIBC 的內容 114
7.3.4 GLIBC 的函式物件 116
7.3.5 C++ 函式庫 118
7.4 多媒體 118
7.5 通訊 119
7.6 系統安全 119
7.7 圖形介面： X 函式庫 119
7.7.1 Xlib 函式庫簡介 119
7.7.2 Xlib 函式庫概觀 122
7.7.3 各式各樣的 Tool Kit 125
7.8 語言 129
7.9 跨平台 129
7.10 其他 129

第八章 作業系統核心與驅動程式 **131**

8.1 作業系統核心 131

8.1.1 簡介 131

8.1.2 Unix 歷史簡介 133

8.1.3 Linux 134

8.1.4 相關網站及參考資料 142

8.1.5 Linux 的延伸與發展 143

8.1.6 BSDi 144

8.2 驅動程式 145

8.2.1 簡介 145

8.2.2 實體裝置的類別 146

第九章 中文方案 **155**

9.1 國際化與本土化 155

9.1.1 區域化資料庫名稱和語系設定 159

9.1.2 GNU/Linux 下的實作方式 161

9.1.3 X Window 的國際化環境 (Xi18n) 164

9.2 中文輸入法 165

9.3 中文輸出 172

9.3.1 簡介 173

9.3.2 圖形輸出格式 : Postscript 與 PDF 174

9.3.3 中文字型的轉換與內嵌 176

9.3.4 中文 Postscript 與 PDF 文件的產生 178

附錄 A Open Public License Draft v0.4 **183**

附錄 B 作者簡介 **187**

第壹篇

自由軟體簡介

第一章

自由軟體簡介

作者：葉平、鄭大慶、胡師賢

以 Linux 爲首的自由軟體熱潮正席捲著全球，不斷地擴大其對全球知識經濟體系與世界各國資訊工業的影響力，許多國際大廠紛紛投入各相關自由軟體的開發工作上。 IBM 的執行長 Lou Gerstner 在 2000 年底宣示： IBM 在一年內將在 Linux 上投資十億美金[1]。海峽對岸更是將 Linux 與自由軟體視爲在這場全球知識經濟的競賽中 —

- 增強國家資訊安全、
- 提升軟體實力、
- 縮短數位落差

的核心基礎建設與技術跳板。身爲資訊大國的台灣自然不能缺席於這股知識經濟的浪潮。

本章是對自由軟體的簡介，我們嘗試回答以下問題：「什麼是自由軟體？」、「自由軟體的優勢在哪裡？」、「自由軟體和傳統商業軟體有何不同？」、「自由軟體的趨勢爲何？」，以做爲之後討論的基礎。

1.1 什麼是自由軟體？

you should think of "free" as in "free speech," not as in "free beer."

— *Richard M. Stallman*[2]

自由軟體前所未有的賦予軟體使用者以下的四種自由[3]：

〔自由０〕使用的自由： 可以不受任何限制地來使用軟體。

〔自由１〕研究的自由： 可以研究軟體運作方式、並使其適合個人需要。

〔自由２〕散佈的自由： 可以自由地複製此軟體並散佈給他人。

〔自由３〕改良的自由： 可以自行改良軟體並散佈改良後的版本以使全體社群受益。

它和傳統商業軟體之間最顯著的差異在於：

第一、自由軟體鼓勵你拷貝。

第二、自由軟體允許你研究、改良。

正因爲自由軟體允許你研究、改良，使得人們得以「站在巨人的肩膀上」，而不必「重新發明輪子」，這種和全球基礎科學研究類似的作法，對於科技的進步有著巨大的影響。

在數以萬計的自由軟體計劃中，Linux 作業系統是最廣爲人知而且影響深遠的自由軟體之一，而今天的 Internet 是以 Sendmail[4]、BIND[5] 等自由軟體爲骨幹架構起來的，此外，絕大部分新開發出來的中央處理器 (CPU) 上的第一個應用軟體開發系統都是自由軟體 Gcc[6]。

1.2 自由軟體的優勢

一言以蔽之，就是控制權。

— *Robert Young, Linux* 紅帽旋風[7]

控制權是自由軟體相對於傳統商業軟體的最大競爭優勢，特別是擁有程式的源碼爲國家在知識經濟時代確保資訊安全的基礎、軟體產業發展與升級的捷徑。

另外從自由軟體的四項自由所衍生出來的

- 可靠、
- 高效能、
- 伸縮性佳、
- 可快速升級、
- 低成本

等競爭優勢，也使得自由軟體所採用的開放程式碼開發方式成爲一種新的標準與趨勢。微軟公司爲了因應此潮流，對於其在資訊家電方面的主力產品 Windows CE 和網路服務 .Net 的程式碼部分，就開始以「微軟程式碼分享授權」(Microsoft Shared Source License)[8] 的方式，希望透過部分程式碼的開放來增加工程師對微軟產品的掌握度，加速客户的產品開發時程與應用品質，由此可見自由軟體的威力。

1.3 自由軟體的運作方式

Linus' Law: Given enough eyeballs, all bugs are shallow.

— Eric S. Raymond[9]

自由軟體是如何能擁有上節所說的各項優點呢？關鍵字是「社群」。網路社群這種數千年首見的組織型態，讓人們在素未謀面的情況下，除了可以透過網際網路上的電子郵件、網站等種種通訊工具進行互動之外，甚至可以協同合作，共同開發程式，這是傳統軟體工程學者無法想像的事，但它的的確確發生了。

這發生的機制 Linus' Law 說的很清楚：只要有夠多的眼睛注視，所有的蟲兒 (bugs) 都很淺顯。也就是說，任何軟體的邏輯錯誤，世界上總有那麼一個人，對他來講找到並修復這個錯誤是易如反掌的，因此，一個人數眾多的社群是自由軟體成功的必要條件。

自由軟體的發展就是在開放網路社群這樣超現代化的運作方式下進行，這和傳統的商業軟體的封閉的運作模式大異其趣。當社群人數到達一定的臨界點之後，Linus' Law 就會成立，這個時候軟體進步的速度飛快，品質大幅提升，吸引了更多人成爲使用者，同時加入社群的人數也會成比例增加。自由軟體的成長循環成形了：社群越大，軟體越好；軟體越好，使用者越多；使用者越多，社群越大（如圖 1.1 所示，見下頁）。

總而言之，社群是 Linux 發展的重要基石，此舉不但推翻了舊時代的軟體發展模式，更重要的是，未來的資訊科技的腳步，一定會像滾雪球一樣的翻騰前進。

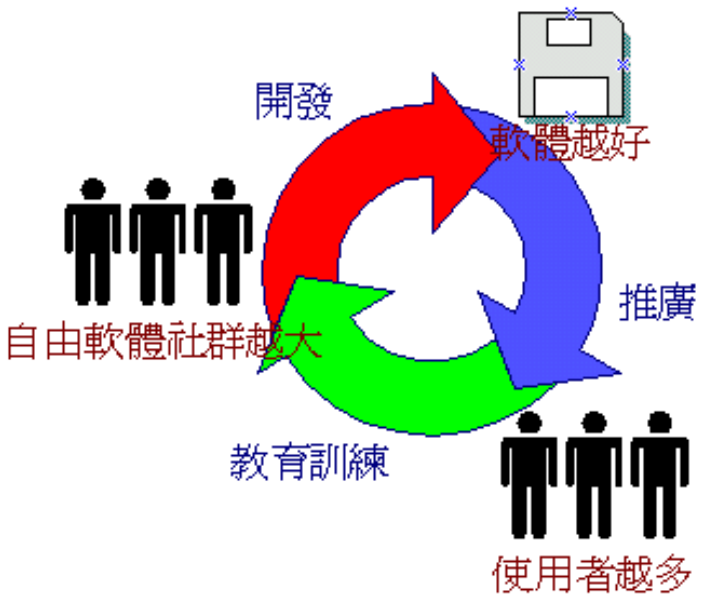


圖 1.1: 自由軟體的成長循環。

1.4 未來的趨勢

Linux will do for software what the Internet did for networks.

— Sam Palmisano at LinuxWorld Expo 2001[10]

未來是網路與通訊的時代，不但是人和人可以利用手機、 PDA、上網機、電腦設備等彼此聯絡，設備與設備之間也將藉由網際網路、 GSM、藍芽、家用無線等通訊方式互相聯繫。 就像以自由軟體架構的 Internet 將所有不同的專屬網路連結起來一樣， Linux 因爲其開放的源碼，將有很大的潛力成爲各種軟體的共同作業系統！

經過長時間的觀察與分析，我們認爲以下三大領域將成爲知識經濟時代資訊產業的重點：

- 資訊家電、
- 伺服器、
- 大型系統。

而自由軟體因其伸縮性、穩定性等優點，在此三大領域均佔有優勢。

軟體自由的概念與科技發展結合、社群與產業的互動、配合程式碼的開放，形成網際網路時代一種新興、成功的程式開發方式。軟體自由協會深信，在下一波知識經濟的循環中，以 Linux 爲首的眾多自由軟體將成爲我國資訊工業的發展過程中不可或缺的角色。

第貳篇

自由軟體發展藍圖

第二章

自由軟體發展藍圖

作者：葉平、胡師賢

擁有穩定、高效能、資訊安全、可靠・・・等特色的 Linux 平台和自由軟體，對於新世紀的資訊產業具有革命性的影響，尤其是其開放源碼的特色，更是台灣資訊科技追上資訊領先國家的最佳利器。 再加上未來資訊家電的普及已成必然之勢，伸縮性佳的 Linux 更是家電跨平台作業系統之最佳選擇。 政府於此關鍵時刻，若能掌握此一契機，台灣未來十年甚至更長遠的軟體產業興盛可期！

我們對二十一世紀 Linux 及其他自由軟體在台灣的願景是：「自由軟體新故鄉」。雖然台灣現在自由軟體的普及率、社群人數和技術深度都落後歐美大國，但是藉由政府的推動，希望軟體自由的精神在台灣落地生根，達成以下幾個目標：

1. 人人會用自由軟體
2. 自由軟體說中文嘛會通
3. 充滿駭客精神的下一代

4. 刺激創意交流的自由環境

5. 開放源碼產業起飛

針對以上之目標，我們提出以下之策略：我們建議政府在以下三大方向發展：

- 支持教育推廣
- 進行基礎建設
- 發展重點領域

建議政府透過政策輔導國內自由軟體社群的成長，加速自由軟體教育、推廣自由軟體使用及開發、並支持自由軟體的本土基礎建設，在重點領域建立社群與產業在科技發展上的連結，厚植國家在知識經濟時代的競爭力。

2.1 支持教育推廣

作者：葉平

從圖 1.1 中可以看出來，自由軟體要能獲得高品質， 前題是有壯大的社群。在台灣，雖然本土社群已存在許久，在技術上也頗有成就，被國際大廠如 Mandrake、 Caldera 等肯定的「 Linux 中文延伸套件[11]」就是社群的傑作。 然而因爲種種原因，自由軟體社群的人數還未達到臨界點，成長循環尚未啓動， 使得社群的工作非常辛苦。我們相信，政府的角色是點燃一把火，把這個循環啓動起來，以後自由軟體在台灣就會滾動起來， 到那時：

- 將會有充足的本土化的自由軟體供大眾和廠商使用，
- 廠商目前「人才不足且不知上哪裡找」的窘境將解除，
- 廠商將可以用熟悉本土化工程的人才作爲跨入全球各地市場的後盾。

所以我們認爲政府推動自由軟體產業的第一要務是提升本土社群的質和量，而要達到這個目的最直接的方法就是舉辦自由軟體教育和推廣活動。

以下是在教育推廣策略下的建議執行方案，並在每個方案之後列出和其相關的目標。

1. 技術之教育：「人人會用自由軟體」

   a) 教育訓練課程：
   輔導教育訓練廠商、資策會、及其他教育訓練機構等開辦自由軟體相關課程，如入門、系統管理、程式設計等等，軟體自由協會樂意在教材準備和教師募集上配合。

   b) 自由軟體研習營：
   和教育部及各大專院校合作，在寒暑假期間舉辦自由軟體營，以集中的方式讓學生接觸自由軟體，並可以在營隊中培養彼此的友誼，對於吸引生力軍進入社群和提升社群的互動性將有很大的幫助。

2. 人才之交流：「刺激創意交流的環境」
   許多創意均來自刺激，而許多刺激均來自人與人的交流。除了在國內促進彼此的交流之外，與國際自由軟體社群交流以增加本土社群之能見度和技術能力也是重點項目。

a) 自由軟體研討會：

舉辦大型的自由軟體研討會，由
政府（如主計處、研考會、教育部 ...等）、
學研界（中研院、各大專院校、資策會、工研院 ...等）、
業界（各廠商如嵌入式、伺服器、大型系統、套件發行 ...等）、
及社群（如軟體自由協會、 CLE計劃、 BSD社群 ...等）
共同參加演講及座談， 並邀請對岸相關人士及國際知名自由軟體人士（如重要自由軟體維護者、 國際大廠高階主管、國際性組織核心人物・・・等等）。 這樣的研討會可促進各界的交流，刺激創意。 在研討會期間也可以舉辦短期教學課程 (tutorial)， 以吸引初學者。

日前舉辦的「開放源碼國際研討會[12]」是現有最接近此項目的活動，應持續支持。

b) 自由軟體讀書會和專題演講系列：

舉辦小型但專注於某單一領域之讀書會 (journal club) 和專題演講 (seminar)， 讓對此領域有興趣的人可以定期聚在一起討論交流， 加速知識的獲取，並增加成員之間的向心力， 而且讀書會在進行一段時間之後，可以將每次成員的報告集結起來， 整理出書，更進一步加速知識的流通。 目前因爲嵌入式 Linux 的需要，已有一些人反應希望能有 Linux kernel 的讀書會；此外軟體自由協會已經開始舉辦物件導向網站平台 zope[13] 的讀書會。

3. 觀念之推廣：

a) 自由軟體展：「人人會用自由軟體」

藉由巡迴的展覽，讓大眾有第一手接觸 Linux 的機會。也可以在展覽中贈送 Linux 介紹光碟並舉辦 Linux InstallFest，讓民眾帶電腦來，由專家爲他安裝 Linux， 需要的話，原有的微軟視

窗的資料和系統都可以完整保留，不必破壞。類似的活動在美國已經行之有年了。

日前舉辦的 Linux Expo[14] 是現有最接近此項目的活動，應持續支持。

b) 駭客精神之宣導和教育：「充滿駭客精神的下一代」

駭客 (hacker) 指的是有好奇心、願意花自己的時間挖出問題本質並解決它、喜歡志願性的互助的人[15]，和大眾媒體上常見的誤解說駭客是專門破壞別人系統的人完全不同。駭客的精神在於勇於挑戰，樂於助人，而 Internet、 Unix、 World Wide Web、 Linux 都是駭客建造出來的傑作。

駭客精神是追根究底的精神，也唯有追根究底，才有可能技術紮根。因此藉由和教育部的合作，在國民教育中放入正確的駭客教材，在高中職甚至更早就可以用 Linux 作爲追根究底的對象，確實做到駭客精神的培養。

c) 自由軟體競賽：「充滿駭客精神的下一代」

和教育部、國家高速電腦中心、中研院．．．等單位以及相關廠商合作， 舉辦自由軟體競賽。一方面可以用自選題的方式鼓勵創意， 另一方面可以和廠商合作，由廠商出指定題和對應的獎金。 軟體作品必須是自由軟體， 並鼓勵使用現有自由軟體組合以彰顯自由軟體的「再利用」精神。 評審應包括業界代表、出資單位、學界代表、社群代表．．等。

2.2 進行基礎建設

作者：葉平

要能有效地推廣自由軟體，一個實用的自由軟體系統和應用軟體是

前提。 此外，本土的開放源碼產業對於一個共同的標準平台需求殷切，因爲唯有這樣的一個平台做爲基礎，才能確保廠商在應用軟體開發上所投注的心血不致於因平台的改變而付諸東流。

綜合以上兩點，我們建議政府提撥經費，由相關單位建立一個

符合國際標準和本土需求的參考平台，

以達到「自由軟體說中文嘛會通」的目標。「符合國際標準」是爲了將全世界的人力納爲我用，讓國際大廠開發出來的平台就是符合台灣本土需求的平台。

現在的時機點對於這個執行項目是非常恰當的，因爲

- 國際標準 Linux Standard Base[16] Common Specification 1.0.0 [17] 剛剛於今年 6 月 29 日公佈，據報導各個國際大廠均將宣佈支援此標準[18] 。
- Linux 國際化的標準 Li18nux[19] 仍有許多待定的項目。

藉由了解 Linux Standard Base 的標準並以國際大廠的版本爲基礎，參加 Li18nux 標準制定並將本土需求提出，將有很好的機會讓複雜的中文字碼和處理問題標準化，成爲國際標準之後，各大廠均將支援，即成爲業界所需的共同平台。 在制定標準的過程中，以一實作平台作爲範例是最好的展現方式，所以我們建議以下的執行項目：

1. 投入人力研究 Linux Standard Base、 Li18nux 和相關標準。
2. 加入 Linux Standard Base 和 Li18nux 組織以參與標準的制定。 在此過程中應邀請對岸相關人士加入，以使繁簡體和各少數民族的需求不致被忽略。

3. 支持一技術團隊實作制定標準過程中的 demo，並成爲參考平台。

有了這個參考平台，套件發行廠商可據以加值或整合成爲 Linux 作業系統套件。不論是誰發行的 Linux 套件，只要是以參考平台爲基礎的，都有共同的應用程式開發介面，使得應用軟體開發者只要開發一次，在任何平台都可以正確地安裝執行，而且因爲符合國際標準，應用軟體可以相對容易地進入全球市場。

2.3 發展重點領域

作者：葉平

自由軟體雖然可以自由取得，但並不代表其中沒有商機：問礦泉水的廠商就知道。 重點是：用自由軟體能不能將顧客要的系統實作出來，而且比其他同類產品更優良， 價格也有競爭力。

答案很清楚：是的！

談到重點領域，不能不勾劃未來的世界。 未來一定是網路的世界，人人上網，家家上網，機機上網。 機機上網，是指任意的電器，如電視、錄影機、冰箱、微波爐、音響、相機、攝影機、掃描器．．．等等，都有上網的能力。 試想像：可以從網路下載節目單、並可透過網路設定錄影時間的錄影機，可以透過網路預視並下載照片的相機，．．．等等。 而要達到機機上網，每機都要有一些軟體來提供 web service 或 browser function。 與其自己開發，然後面對一般 web server / browser 功能的升級所帶來的壓力，和滿足不斷出現的新協定的困難，不如以經過社群千錘百煉的自由軟體爲本，經過適當的瘦身來完成。 如此的軟體較自行開發更能達到商用軟體的需求。

在產業部分，我們認爲傳統 PC 產業，也就是製造、販售「廣泛用途 PC」的產業可能已經開始走下坡，但是以 PC 或類似結構做「專屬用途」的市場才正要開始發展，如伺服器、資訊家電和大型系統等。而 Linux 等自由軟體的伸縮性、控制權在握等特色，使得自由軟體比傳統商業軟體在這三個領域更可以揮灑自如。

1. 大型系統

   不論在科學研究、工程計算或是新一代的生化領域，超級電腦所供應的強大運算能力一直是一個相當重要的工具，但其昂貴的價格卻又使得此項設備的擁有與使用無法普及。運用 Linux 與相關的計算工具軟體可使超級電腦的建置成本大幅降低，並提升其擴充性。早在數年前美國國家實驗室就開始運用這項技術成功地省下數百萬美金的設備建置費用。龐大市場所隱含的商機也使得傳統超級電腦大廠如 Cray 和 IBM 等公司趨之若鶩。

   除了超級電腦之外，Linux 在其他大型系統也有無窮的潛力，如銀行商務系統、電信帳務系統、售點系統、．．．等等，現在全世界最強大的搜尋引擎 Google 就是用超過 6,000 台 Linux PC 架設的。

   參考：

   (1) 美國 Sandia 國家實驗室的 CPlant 計畫[20]。

   (2) 用在探油的 Linux 超級電腦[21]。

   (3) Cray 開始開發以 Linux 爲作業系統的超級電腦[22]。

   (4) 用 Linux 的售點系統[23]。

2. 伺服器

   網站、電子郵件伺服器等網路設備的需求隨著網際網路一同不斷地成長。運用 Linux 與相關自由軟體所建立的眾網路伺服器也在社群

的努力下，使用最先進的技術從而具有完整的功能與穩定運行的品質。 提供網站服務的 Apache[24] 伺服器就佔據了全球 60% 以上的使用率，電子郵件伺服器 Sendmail 與網域名稱伺服器 Bind 的佔有率更分別達到了 75% 與 90%。

參考：

(1) netcraft [25]

(2) activemedia-guide [26]

(3) zdnet [27]

3. 資訊家電

PC 等資訊產品向爲台灣出口大宗，但隨著 PC 市場需求的趨緩，資訊應用產品 IA 成爲台灣是否能順利步入後 PC 時代的重要關鍵。嵌入式作業系統是 IA 軟體運作的基礎， Linux 以直逼 Unix 系統的穩定性、其開放程式碼所提供高度的調整性與安全性等優點，成爲嵌入式作業系統的最佳候選者。嵌入式 Linux 不但被眾多資訊廠商作爲著力的重點，更被視爲台灣提昇軟體研發水準，擴大產值的一個重要機會點。

參考：

各相關報導[28, 29, 30, 31].

我們建議政府斥資支持以自由軟體在以上三大領域研究開發的努力，以使台灣資訊產業在二十一世紀進入一個新高峰！

第參篇

自由軟體元件細圖

第三章

自由軟體與微軟視窗平台的對照表

作者：鄭大慶、吳鴻煦、李柏峰、胡師賢、葉平

3.1 簡介

作者：鄭大慶

誠如第一章的介紹，自由軟體的高效能、高穩定性與低成本的特色，使得它成爲網路運作的最佳選擇。在網際網路上的三大伺服器領域：網域名稱伺服器、電子郵件伺服器、網站伺服器方面，Apache、Sendmail、BIND 這些自由軟體分別高據全部市場佔有率的 60%、75%、90%。所謂「大者恆大」，這些軟體的高使用量也是其品質不斷持續成長的來源。

在目前全球性的 e 化趨勢中，不論政府、企業或其他組織，都企盼能夠掌握網際網路的技術來增加核心競爭力。自由軟體構成了網際網路上基礎建設的主幹，當然是這股 e 化洪流中的不可或缺的要角。而且，其開放源碼的特性，可以賦予使用者或是程式開發人員高度的掌控權力，對軟體的研究發展提供莫大的助益，這是微軟等傳統商業軟體所無

法比擬的。特別是在台灣資訊工業當前的焦點－嵌入式系統的開發上，具有不容取代的價值。

本章的主要目的就在於站在實用的角度，就 e 化解決方案、軟體程式開發及嵌入式系統等方面，提供各種自由軟體運用的解決方案，並與微軟等商業軟體的付費方案比較，讓使用者多一個替代方案的選擇機會。本章將以比較表的方式來方便本書使用者的查詢，觀念性地說明各解決方案的主要內容。至於各自由軟體的相關技術更進一步的說明，會在隨後的各章節展開。

3.2 解決方案之比較

作者：吳鴻煦、鄭大慶

Linux 在商業應用系統之實用解決方案，包含網路應用、商業應用、資訊系統及教育應用。

應用系統	自由軟體解決方案	商業解決方案
網站架設	Apache + Linux†	IIS + Windows‡
網頁開發	Apache + PHP + Linux	IIS + ASP + Windows
資料庫	MySQL/PostgreSQL	Oracle, MSSQL
電子商務應用	LAMP⊗	IIS + ASP + MSSQL + Windows
路由器	Linux + gated/zebra	Cisco router
檔案伺服器	Linux + Samba	Windows
列印伺服器	Linux + Samba	Windows
防火牆	Linux + opensourcefirewall	Checkpoint Firewall + Windows
Proxy 伺服器	Squid proxy server + Linux	MS Proxy Server + Windows
目錄服務伺服器	Openldap server + Linux	MS Active Directory Server + Windows
電子郵件伺服器	Sendmail/postfix + Linux	MS Exchange Server + Windows
名稱伺服器	Bind + Linux	MS DNS + Windows
DHCP	DHCP server + Linux	DHCP Server + Windows
文書處理軟體	openoffice	MS Office
應用伺服器	Jboss/Jakarta-tomcat	MS MTS

† Windows: 指微軟 Windows NT 或 Windows 2000。

‡ Linux: 泛指各種自由軟體作業系統如 GNU/Linux、 FreeBSD 等，這裡爲了節省空間而以 Linux 表示之。

⊗ LAMP: 是 Linux + Apache + MySQL + PHP/Perl/Python 的標準縮寫，是當今最受歡迎的自由軟體網站開發平台。

- 網站架設

  Apache 是全世界市場佔有率最高的網站伺服器。 除提供基本的 http 通訊協定外，對於虛擬主機、網路安全傳輸規格 SSL 及 PHP

等程式模組的擴充功能都有支援。在各方面的測試下所展現的高穩定性與高效能使得它成爲網站伺服器的首選。目前可以在 Linux、Windows、Solaris 等多種平台上執行。

- 網頁開發

  PHP 是動態網頁的支援模組，提供程式師開發瀏覽器界面 (WEB-BASE) 系統的功能。擴充性強，幾乎支援包括 Oracle 在內的所有資料庫的 SQL 查詢，與包括 LDAP、POP3 在內的多種通訊協定，與 XML/XSL、PDF、FLASH 等多種文件處理。由於其易學、功能完整的特性，全世界使用 PHP 的網站已經超過七百萬個。

- 資料庫

  MySQL 是 SQL 關連式資料庫，由於執行效能與穩定性高，操作簡易所以使用者眾多。PostgreSQL 則是物件關連式 (object-relational) 資料庫，功能完整，支援 SQL92/SQL93 資料庫查詢規格。這兩種資料庫都是資料庫網站建置的最常見的選擇。

- e 化應用程式開發平台

  網站伺服器 Apache、資料庫 MySQL/PostgreSQL 和 PHP 程式模組的連結，形成一個網站資料庫的開發平台。目前平台上已開發的各種應用程式，可以提供入口網站 (Portal)、知識管理 (KM)、客户關係管理 (CRM)等服務，是目前相當流行的一種 e 化方式。

- 應用伺服器

  Java 的物件特性非常適用於大型的應用系統的開發，Java Servlet 與 JSP 都是網路伺服器上常用的 Java 技術。Apache 團隊所開發的 Jboss 及 Jakarta-tomcat 等軟體，就是支援這些技術的引擎，提供 Java 應用程式執行所需要的環境。目前雖然還在發展中，但極被重視，已擊敗 IBM 與 SUN 獲得多項獎項。

3.3 開發系統的比較

作者：李柏峰

圖 3.1 是 Gnome 系列程式庫的結構圖，不過一般而言圖形介面的應用程式，不論是否在 Gnome 系列上開發，結構也是大同小異。

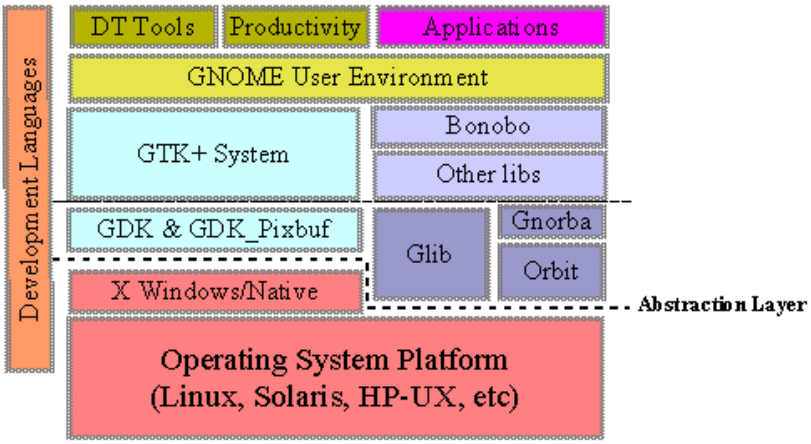


圖 3.1: Gnome 系列程式庫的結構圖。 來源：[?]。

以下是在微軟平台上和自由軟體領域中開發系統的比較表：

	標準	自由軟體	微軟平台
視覺開發環境		glade, KDevelop	Visual C++
圖形介面描述		libglade, XML-UI/Kparts	
元件系統		Bonobo, Kparts	OLE2
資料庫連結	ODBC/SQL	Gnome-DB/libgda, KDE-DB	ADO.NET
物件系統		Gnorba/ORBit, DCOP	COM
分散式物件系統	RMI/CORBA	OAF/ORBit, kxmlrpcd	DCOM
高階圖形介面程式庫		gtk−−/gtk+, Qt	MFC
低階圖形介面程式庫		gdk/XLib	Win32 GDI(?)
C 語言程式庫	UNIX98,POSIX	glibc	MSVCRT.DLL
核心	POSIX	linux, Hurd, BSD	KERNEL32.DLL

- The GNU C library (glibc[32])

  大體而言 GNU C 函式庫 (glibc) 支持 ISO C 和 POSIX 標準，他們也設法支持受歡迎變體 Unix 的特徵 (包含 BSD 和 System V)，它甚至追隨了最近的標準 (ISO C 99, POSIX.1c, POSIX.1j, POSIX.1d, Unix98)。

- The GNU Hurd (hurd[33])

  GNU Hurd 是 GNU 計劃中， Unix 核心 (如 Linux) 的替代品。Hurd 是指一組執行於 Mach 微核心之上的伺服器 , 他們提供了檔案系統、網路協定、檔案存取控制、・・・等等原來 UNIX 核心所提供的功能。

- The GIMP Toolkit (gtk+[146])

  GTK+ 是一個用來建立圖形化使用者介面的跨平台工具組，目前主要是架在 X Window System 之上。它由 C 語言所撰寫，但有著物件導向的結構，由下列的元件所組成：

  – GLib: 提供許多資料類型、巨集、類型轉換、字串工具、詞法分析的函式。

  – GDK: 低階視窗函數的 wrapper (以保障 gtk 的可攜性)。

  – GTK: widget collection

  目前穩定的版本是 1.2，有完整的 Ada、 C++、 Perl、 Python 支援，發展中版本是 1.3。 GTK+ 2.0 承諾會移植到 MS Windows、Linux Framebuffer、 BeOS。將由 Glib、 GObject、 Atk、 Pango、GdkPixbuf、 GDK、 GTK 所組成。

- The Object Activation Framework (OAF[35])

  OAF 是個用來定位並執行 CORBA 伺服器的系統。

- The Bonobo component system (Bonobo[36])

  Bonobo 是一組 CORBA 介面，這組介面定義了軟體元件與複合文件的互動。這組 CORBA 介面是獨立的，並且與 GNOME、 X 視窗系統、或任何 UNIX 無關。

- The ORBit Object Request Broker (ORBit[37])

  ORBit 是個符合 CORBA 2.2 的物件訊息仲裁者 (Object Request Broker)， ORBit 有著成熟的 C 和 Perl 支援 (也有部分的 C++, Lisp, Pascal, Python, Ruby, and TCL 支援)， 並支援 可攜型物件轉接模組 (Portable Object Adapter, POA)、動態介面 (Dynamic Invocation Interface, DII; Dynamic Skeleton Interface, DSI; TypeCode, Any)、 實作庫 (IR, Implementation Repository)、 網際網路 ORB交換協定 (Internet Inter-ORB Protocol， IIOP)。

- Glade[38]

  Glade 是個支援 gtk+ 的圖形介面建造器。以視覺化的方式產生圖形介面後，它能自動產生 C 原始碼， Glade 會將此圖形介面暫存一個於符合 XML 標準的檔案 (*.glade)。 藉由外部工具的幫助，它也可以產生 C++、 Ada95、 Python & Perl 原始碼。

- Libglade[39]

  Libglade 是個函式庫，其功能類似 Glade，差別只是 Glade 會產生 C 原始碼， 可用來編譯，但 libglade 可以利用 glade 檔來動態的建立圖形介面。所以你可以利用 libglade 改變你的程式的外觀，但不需要重新編譯你的程式。

- Libgda/Gnome-DB[40]

  GDA (GNU Data Access) 嘗試提供一致的界面，以用來存取不同種

類的資料 (資料庫 , 資訊伺服器 , 電子郵件 ...等等)。它提供了存取資料的完整架構， GDA 是一組 CORBA IDL 所定義的介面，並儘可能的一般化 (generic)。因此，我們幾乎可以經由 GDA 來存取任何種類的資料。 GDA 函式庫 (libgda) 實作了 GDA 介面，並提供了伺服端與客户端用來存取 GDA 架構的函式，它也提供了許多工具以幫助你管理資料。

Gnome-DB 提供 Gnome 計劃中資料存取的架構。 Gnome-DB 是基於 libgda 所開發的，但是 libgda 完全與 Gnome 無關，所以可以用 libgda 開發其它程式。

範例 :

GDA client	
GDA client library	(a CORBA client library + GDA widget library)
GDA provider	(1. CORBA objects implementing the GDA CORBA interfaces)
	(2. shared library provider)

- KDE libraries[41]

  K Desktop Environment 開發所需之程式庫。

- KParts[42]

  KDE 的複合文件 (元件)系統稱作 Kparts. 一個 KDE 元件稱爲 part，它包含了三部分： widget (window gadget, 視窗配件)、widget 功能以及使用者介面。

- XML-UI/Kparts[43]

  程式師可以使用 XML-UI 描述 KParts 的圖形介面。

- Qt[44]

Qt 是一個跨平台的 C++ 圖形介面發展架構，並內附視覺化的開發環境，以 QPL(Q Public License) 和 GPL 分別發行，在 MS Windows 平台則允許個人免費使用。

- Desktop COmmunications Protocol (DCOP)[45]

  DCOP 是個簡單的 IPC/RPC 機制，建構於 ICE 協定與 Qt 函式庫之上，而 ICE (Inter Client Exchange protocol) 是 X 視窗系統的一部份。 DCOP/ICE 可提供 KDE 桌面環境中元件互相溝通的機制。

- XML-RPC[46]

  一個跨平台的遠端程序呼叫的規格，以 http 做爲傳輸的協定，以 XML 做爲內容的格式。

- KDE XmlRpc Daemon (kxmlrpcd)[47]

  kxmlrpcd 本質上是個 web 伺服器，並負責將外來的 XML-RPC 解譯爲 DCOP 呼叫，所以我們幾乎可以在任何平台，用任何語言，以 XML-RPC 呼叫 KDE 元件。

3.4 嵌入式系統

作者：胡師賢

Linux 除了在標準硬體平台上展現其廣泛的商業應用價值之外，進入後 PC 時代後更是內嵌式系統（ Embedded System）的發展所繫，在以提昇產業競爭力爲目標的研發與創新過程中，扮演極爲重要的關鍵性角色。所謂的內嵌式系統指的就是爲了提高產品的穩定性、可攜性，或者是降低產品成本、增加特定功能等不同的目的，對一般的標準系統進行修改以符合特殊設計上的需求。這種內嵌式的做法在硬體方面靠的主要就是被稱爲系統單晶片（ SOC - System On a Chip）的半導體整合技術，而內嵌式作業系統則是軟體方面的技術核心。自由軟體的可靠、高效能、伸縮性、低成本，以及最重要的握有控制權等幾項特點，使得 Linux 在內嵌式作業系統上站在非常有利的位置。以下就針對 Linux 在內嵌式作業系統與商業解決方案中最主要的 WinCE 做一比較。

	Linux	WinCE
伸縮性（程式碼）	自由取得	只開放給稱爲 WESSA 的特定對象
跨平台（處理器）	ARM、 MIPS、 PowerPC、 SH、 x86、 Alpha、 IA64、 M68K 以及更多	ARM、 MIPS、 PowerPC、 SH、 x86
功能	TBI†	TBI
性能	TBI	TBI
開發時間	TBI	TBI
成本	自由使用	必須支付數美元至數十美元不等的授權費用

†: TBI: to be investigated.

3.5 自由軟體和其他之實測比較

作者：葉平

自由軟體的特色是全球聯合開發，而這樣開發出來的自由軟體，和其他軟體在實際的測試比較下，得出以下的優點[48]：

- 可靠 (Reliable)：
  有關自由軟體比較可靠的說法有很多，但再多論述也比不上實測，以下我們列舉幾個實測結果：

  1. Sm@rt Partner 於 1999/11 公佈 Windows NT、 OpenLinux 和 Red Hat Linux 在完全相同的硬體上提供網路服務十個月的可靠度評比，發現 NT 平均六個禮拜當機一次，每次平均要花 30 分鐘修復，但是兩種 Linux 連一次當機也沒有[49]。
  2. 瑞士的網路效能監視專家 SysControl AG 公司於 2000/02 公佈了他們對瑞士最受歡迎的一百大網站的可靠度評比結果。SysControl AG 花了三個月的時間從四個不同的地方每五分鐘探測一次每個受測網站，每個月的當機時間結果如下[50]：

當機時間	Apache	IIS	Netscape	Andere
九月	5.21	10.41	3.85	8.72
十月	2.66	8.39	2.80	12.05
十一月	1.83	14.28	3.39	6.85
平均	3.23	11.03	3.35	9.21

     可以看出：用微軟 IIS 的網站當機時間是 Apache 的二倍以上[51]。

- 高效能 (High Performance)：
  這裡引用兩個實測結果：

1. Sm@rt Partner 在 2000 年測試了網站伺服器的效能： Microsoft IIS on NT 4.0、 Apache on Red Hat 5.2、 SuSE 5.3、 Caldera 1.3。結果發現：三種 Linux 中效能最差的 SuSE，還比 IIS 快 16%。
   詳見 [52]。
2. SysAdmin 期刊在 2001/07 發表了他們對 Linux、 Solaris、 FreeBSD 和 Windows 2000 的網路效能測試評比，結果是 Linux 領先第二名的 Solaris 達 35% 之多。
   詳見 [53]。

- 安全性 (Security)：
  眞正安全的系統，是經過千錘百鍊的系統，而不是把規格藏起來的系統。自由軟體經過千千萬萬個使用者在各種不同環境下的使用，開放出來的程式也經過許多社群同好的檢驗，即使有安全上的漏洞也以網路時代的速度修補好了，因此自由軟體系統是最安全的系統。若是封閉系統的話，使用者還得要擔心看不見的程式內部有沒有藏著後門或是不爲人知的漏洞。

  有數據可以支持這個說法：在 2000/01/17， SecurityPortal 公佈了 Red Hat、 Microsoft 和 Sun 三家公司的作業系統在 1999 年全年的安全弱點個數和修復的速度，以下是比較表：

公司	弱點總數	弱點曝光天數	平均弱點曝光天數
Red Hat	31	348	11.23
Microsoft	61	982	16.10
Sun	8	716	89.50

  可見： (1) Linux 的弱點較少。 (2) Linux 的弱點曝光期間較短。若把社群開發出修復程式到 Red Hat 公佈之間的延遲計入，曝光期會更短。這顯示了 Linux 比較安全。原文詳見 [54]。

- 伸縮性 (Scalability)：
  伸縮性講的是軟體或解決方案在問題規模變大或變小的時候能否繼續正常運作。許多商業應用或頂尖的研究計劃都要面對這個問題。最簡單的例子，比方說網站伺服器：當用户只有 50 個人的時候，一台跑任何網站伺服軟體的 PC 都處理得了，但是當用户激增到 50,000 人的時候，怎麼辦？最簡單的想法是增加 PC，但是若原有的軟體不能在多 PC 的環境下正常工作，那就出問題了！

  許多自由軟體都是跨平台的，甚至連作業系統層次的自由軟體如 Linux 都可以支援十幾種不同的硬體，如 Intel x86、 ARM、 Alpha、 Sparc · · · 等等，配合 Linux 在網路上的優勢，這使其伸縮性大增。

  往大的方向最直接的例子莫過於現在全世界功能最強大的搜尋引擎 Google，它是由超過六千台的一般 Linux PC 所架成，儲存全世界超過十三億個網頁，每天提供超過一億次的搜尋服務[55]。

  往小的方向，小到如 Compaq 的 iPAQ PDA 也是可以執行 Linux，更小的如 IBM 的 Linux 原型手錶。可見自由軟體的伸縮性十足。

- 快速升級：
  一般常用軟體通常要數個月甚或一年以上才會有新版，且升級需付費。 自由軟體通常每週都有修正版，半年左右即有升級版，且升級爲免費。

- 低成本：
  不用付昂貴的授權費用，可以自由取得、自由複製。

從以上的實例和邏輯來看，可靠、高效能、安全、伸縮性佳、可快速升級、低成本又握有自主權的自由軟體對軟體使用者實在是最好的選擇。

第四章

網路應用 (Network Applications)

作者：吳鴻煦、謝東翰

在 Internet 上， Linux 扮演非常重要的角色，尤其在網路的部分，不但提供 TCP/IP IPV4 及 IPV6 的完整實作， 更有相當多免費的網路軟體。最具代表性的網路應用軟體有 apache、 sendmail、 samba 等等相當流行及實用的軟體。

4.1 網路應用伺服系統

1. openldap [56]
2. enhydra(XML application server) [57]
3. jakarta tomcat(servlet container)支援 java servlet 2.2、 2.3 規格
4. jboss(EJB container)提供快速強大的 J2EE 環境 [58]
5. samba 檔案及列印伺服器系統可取代 Windows NT / 2000 [59]

4.2 網路共用程式庫

- libpcap [60]

  這是一個小型的函式庫，它提供了以 BSD 封包過濾器 (BPF) 爲基礎的封包過濾機制。最著名的是， tcpdump 需要它才能運作，而且還有一個 perl 的模組 (仍然處於 beta 版) 也能使用它。簡單地來說，如果您想要寫一個您自己的網路流量分析器，則這就是您的工作起點。

- libnet [61]

  原始封包的產生與傳送函式庫。

- libnids [62]

  IP 拆解與 TCP 封包片段重組函式庫。

- Libping [63]

  一個小型的 C 函式庫用以產生 ICMP_ECHO 呼叫需求。

- iSSL [64]

  iSSL (independant Secure Sockets Layer) 是一個抽像的編碼 API，它使用 RSA 與 AES 演算法，以在網路連線的兩個通訊端之間，建立類似 SSL、安全加密的連線，同時包含了通訊編碼鑰匙的產生以及公共鑰匙的交換等。

- GNet [65]

  GNet 是一個簡單的網路函式庫。它是物件導向的，由 C 語言寫成，同時也以 GLib 爲基礎建構而成。其設計的目的是小、快、容易使用且容易移植。 GNet 支援抽像的網路位址、 TCP、 UDP、

IP Multicast、非同步 DNS 查詢、以及 TCP 連線。它還內附了說明文件以及範例。

- Cyrus SASL [66]

  Cyrus SASL 函式庫是一個可以很容易在任何客户端與伺服端應用程式上，整合安全網路認證的一般性函式庫。它支援標準平面文字的認證方式，以及加密的認證方式，還有 KERBEROS_V4 和 GSSAPI Kerberos 方式等。 SASL 協定體系是使用在 SMTP、IMAP、 ACAP、 LDAP、以及其他標準的協定上。

- Libsocketcpp [67]

  libsocketcpp 提供了容易使用的 C++ class，可以讓我們經由呼叫一些簡單的程序以實作簡單的 UDP 以及 TCP。

4.3 網際網路伺服器

- squid
- apache
- samba(print server, file server)
- NFS(nfsv2, nfsv3)
- NCP(Novell Server)
- 名稱伺服器 Domain Name Server(bind)
- 電子郵件伺服器 Mail Service(sendmail, postfix, exim)
- 新聞伺服器 NEWS Server(inn)

- 目錄服務伺服器 LDAP Server(openldap)
- 通信伺服器
- 傳眞伺服器 [68]

  HylaFAX 是一個在 UNIX 系統下的通訊傳眞系統。它支援了發送傳眞、接受傳眞、優先取回傳眞、 modem 的透明化資料分享使用，以及傳送 alpha-numeric 文件頁。這個軟體是以客户端 –伺服器架構所構成。傳眞用 modem 可以安置於網路上的一部機器，而客户端可以在任何可連上該 modem 伺服器的機器上交付遠端傳眞工作。它還內含了一個存取控制機制以控制那一部機器上的那一個使用者可以使用傳眞伺服器。

- RAS server
- PBX(openh323)
- BIND [5]
- kernel mode file system

4.4 網路檔案系統

- Global File System [69]

  全域檔案系統 (Global File System (GFS)) 是一個在 Linux 上的 64 位元硬碟分享叢集系統的檔案系統。每一個 GFS 叢集系統上的運算單元藉由 Fibre 管道或分享式 SCSI 裝置共用一個儲存設備。檔案系統在每個運算單元上就好像是本機的檔案系統，而且 GFS 可以在整個叢集系統間同步存取檔案。 GFS 是完全對稱的，意思就是所有的運算單元都是完全平等的，且不會有可能形成瓶頸的伺服器或單

一點的錯誤。 GFS 使用了讀寫快取以維持完整的 UNIX 作業系統特性。 GFS 還支援日誌功能、自客户端的錯誤修復，以及其他許多特色。

- NFSD
- ENBD

4.5 網路管理

- Qos(Kernel-based CBQ)
- Bandwidth/Traffic Management
- Network Management Station(ucd-snmp)
- zebra [70]

  GNU Zebra 管理 IPv4 與 IPv6 的封包路由 (routing) 協定。它支援 RFC1771 文件 (A Border Gateway Protocol 4) 中所描述的 BGP-4 協定，以及 BGP-4+、 RIPv1、 RIPv2、 RIPng、 OSPFv2 和 OSPFv3 等協定。 GNU Zebra 具備了不錯的客户端介面，故使用者可以動態改變其設定組態。

- GateD [71]

  GateD 是一個路由伺服器 (router)，它支援大部分的常見 Unicast 路由協定，包括 RIP, OSPF, 以及 Router Discovery。其 ipv6 的版本還支援了 RIPng 與 BGP-4+。

4.6 網路安全工具

- SSL packages(openssl)[72]
- SSH packages(openssh)[73]
- NAT(Kernel-Based iptable)[74]
- Firewall(Kernel-Based iptable)
- VPN
- IPSec(freewan)
- network sniffer(tcpdump, ethereal)
- network intrusion detection(snort)

4.7 防火牆

- iptables
- opensourcefirewall
- sock4/5 server

第五章

資料管理 (Data Management)

作者：吳鴻煦、李健秋、謝東翰

5.1 資料庫 (Database)

作者：吳鴻煦

5.1.1 MySQL 資料庫

什麼是 MySQL？

（註：此節翻譯自 MySQL 線上手冊，因此文中的「我們」指的是 MySQL 開發團隊）

MySQL 是一個快速、多線性 (multithread)、多使用者和強壯的 SQL 資料庫伺服器。尤其在 select/insert/update 等基本 SQL 指令之執行效率非常快速，比商業之 MSSQL 及 Oracle 資料庫等更快，更重要的是 MySQL 是免費的。同時支援中文 Big5 碼 。

MySQL 是一個眞正的多使用者、多線性 SQL 資料庫伺服器。

SQL（結構化查詢語言）是世界上最流行的和標準化的資料庫語言。MySQL 是以一個主從式架構的實現，它由一個伺服程式 mysqld 和很多不同的用户端程式庫組成。

SQL 是一種標準化的語言，它使得存儲、更新和存取信息更容易。例如，你能用 SQL 語言爲一個網站檢索產品信息及存儲顧客信息，同時 MySQL 也足夠快和靈活以允許你存儲記錄文件和圖像。

MySQL 主要目標是快速、穩固和易用。最初是因爲我們需要這樣一個 SQL 服務器，它能處理與任何可不昂貴硬件平台上提供資料庫的廠家在一個數量級上的大型資料庫，但速度更快， MySQL 就開發出來。自 1996 年以來，我們一直都在使用 MySQL，其環境有超過 40 個資料庫，包含 10,000 個表，其中 500 多個表超過七百萬行，這大約有 100 GB 的關鍵應用資料。

MySQL 建立的基礎是業已用在高要求的生產環境多年的一套實用例程。盡管 MySQL 仍在開發中，但它已經提供一個豐富和極其有用的功能集。

MySQL 的官方發音是 “My Ess Que Ell” (不是 MY-SEQUEL)。

MySQL 歷史

Tcx 公司曾經開始打算利用 mSQL 用我們自己的快速底層 (ISAM) 實用程序連接我們的資料庫表，然而，在一些測試以後我們得到出結論：mSQL 對我們的需求來説不夠快速和靈活。這產生了一個連接我們資料庫的新 SQL 介面，但它幾乎有與 mSQL 相同的應用程式介面。選擇這個應用程式介面以便利第三者的移植。

MySQL 名字的由來不是非常清楚。我們的目錄和很多的程式庫有前碼 “my” 已超過 10 年歷史，然而， Monty 的女兒 (年輕幾歲的)也被命

名 “My”。因此其中哪一個原因給 MySQL 起了這個名字仍然是一個謎。

MySQL 的主要特性

MySQL 一些重要的特性：

- 使用核心線性的完全多線性。這意味著它能很容易地利用多 CPU（如果有）。
- C 、 C++、 Eiffel 、 Java、 Perl、 PHP、 Python、和 TCL API。見 20 MySQL 用户端工具和 API。
- 可運行在不同的平台上，見 4.2 MySQL 支援的操作系統。
- 多種列類型： 1、 2、 3、 4、和 8 字節長度的有符號 /無符號整數、 FLOAT、 DOUBLE、 CHAR、 VARCHAR、 TEXT、 BLOB、 DATE、 TIME、 DATETIME、 TIMESTAMP、 YEAR、 SET 和 ENUM 類型。
- 利用一個最佳化的一遍掃描多重聯結（ one-sweep multi-join）非常快速地進行聯結 (join)。
- 在查詢的 SELECT 和 WHERE 部分支援全部運算子和函數，例如：

```
mysql> SELECT CONCAT(first_name, " ", last_name) FROM tbl_name
       WHERE income/dependents > 10000 AND age > 30;
```

- 通過一個高度最佳化的類程式庫實現 SQL 函數庫並且像他們能達到的一樣快速，通常在查詢初始化後不應該有任何内存分配。

- 全面支援 SQL 的 GROUP BY 和 ORDER BY 子句，支援聚合函數 (COUNT()、 COUNT(DISTINCT)、 AVG()、 STD()、 SUM()、 MAX() 和 MIN())。
- 支援 ANSI SQL 的 LEFT OUTER JOIN 和 ODBC 語法。
- 你可以在同一查詢中混用來自不同資料庫的表。
- 一個非常靈活且安全的權限和密碼系統，並且它允許根據主機的認証。 密碼是安全的，因爲當與一個服務器連接時，所有的密碼傳送被加密。
- ODBC for Win32。所有的 ODBC 2.5 函數和其他許多函數。例如，你可以用 Access 連接你的 MySQL 服務器。
- 具備索引壓縮的快速 B 樹表。
- 每個表允許有 16 個索引。每個索引可以由 1 16 個列或列的一部分組成。最大索引長度是 256 個字節 (在編譯 MySQL 時，它可以改變)。
- 一個索引可以使用一個 CHAR 或 VARCHAR 字段的前碼。
- 固定長度和可變長度記錄。
- 用作臨時表的內存散列表。
- 大資料庫處理。我們正在對某些包含 50,000,000 個記錄的資料庫使用 MySQL。
- 所有列都有缺省值，你可以用 INSERT 插入一個表列的子集，那些沒用明確給定值的列設置爲他們的缺省值。
- 爲了可移植性使用 GNU Automake , Autoconf 和 libtool。

- 用 C 和 C++ 編寫，並用大量不同的編譯器測試。
- 一個非常快速的線性的記憶體分配系統。
- 沒有內存漏洞。用一個商用內存漏洞監測程序測試過 (purify)。
- 包括 myisamchk，一個檢查、最佳化和修復資料庫表的快速實用程序。
- 全面支援 ISO-8859-1 Latin1 字符集。例如，斯堪的納維亞的字符 å, ä and ö 在表和列名字被允許。
- 所有資料以 ISO-8859-1 Latin1 格式保存。所有正常的字符串比較是忽略大小寫的。
- 根據 ISO-8859-1 Latin1 字符集進行排序 (目前瑞典語的方式)。通過在源程式中增加排序順序數組可以改變它。爲了理解一個更高級的排序例子，看一看捷克語的排序程式。 MySQL支援可在編譯時指定的很多不同的字符集。
- 表和列的別名符合 SQL92 標準。
- DELETE、 INSERT、 REPLACE 和 UPDATE 返回有多少行被改變 (受影響)。
- 函數名不會與表或列名衝突。
- 服務器能爲用户端提供多種語言的出錯消息。
- 用户端端使用 TCP/IP 連接或 Unix 套接字（ socket）或 NT 下的命名管道連接 MySQL。
- MySQL 特有的 SHOW 命令可用來檢索資料庫、表和索引的信息，EXPLAIN 命令可用來確定最佳化器如何解決一個查詢。

MySQL的穩定性

對 TcX， MySQL 在我們自 1996 中期開始的計劃中運行沒有發生任何問題。當 MySQL 被更公開地發布時，我們注意到了有一些“未測試程式”片斷很快地被不同于我們的查詢方式的新使用者發現。每個新版本比前一個都有更少的可移植性問題 (盡管每個發行有許多新功能)，並且我們希望有可能把下一個版本之一標記爲“穩定”的。

每個 MySQL 的發行都是可用的，並且只有當使用者使用從“灰色地帶”來的程式時才有問題，當然，不知情的使用者不能知道灰色地帶是什麼；本小節嘗試揭示我們目前已知的東西。這裡的描述涉及 MySQL 3.22.x 版本。

MySQL 以多層結構和不同的獨立模塊編寫，這些模塊列舉在下面以表明它們中的每一個是如何很好地被測試過：

- ISAM 表處理器 — 穩定

  它管理所有在 MySQL 3.22 和早期版本中的資料的存儲和檢索。在所有 MySQL 版本中，程式中已經沒有一個單獨（報告的）錯誤。得到一個損壞的資料庫表的唯一已知方法是在一個更新中途殺死服務器，即使這樣也不大可能破壞任何資料而不能挽救，因爲所有資料在每個查詢之間被倒入 (flush)到硬碟，而且從來沒有一個有關於 MySQL 中的錯誤而丟失資料的錯誤報告。

- MyISAM 表處理器 — 穩定

  這是 MySQL 3.23 的新功能，它大部分是根據 ISAM 表程式但有很多新的有用的功能。

- 語法處理器和詞法分析器 — 穩定

  很長時間沒有一個在這個系統中的錯誤報告。

- C 用户端程式 — 穩定

  沒有已知的問題。在早期的 3.20 版本中，在發送 /接收緩沖器的大小上有一些限制。 3.21.x 後，現在緩沖器的大小是動態的，可到一個 24M 的缺省值。

- 標準用户端程序 — 穩定

  這些包括 mysql、 mysqladmin 和 mysqlshow、 mysqldump 及 mysqlimport。

- 基本結構式查詢語言 — 穩定

  基本 SQL 函數系統、字符串類和動態内存處理，本系統中未見單獨報告的錯誤。

- 查詢最佳化程序 — 穩定

- 範圍最佳化程序 — 穩定

- Join 最佳化器 — 穩定

- 鎖定 — Gamma

  這是非常依賴于系統的，在某些系統上，用標準操作系統鎖定 (fcntl())有很大問題，在這些情況下，你應該用選項 `--skip-locking` 運行 MySQL 守護程序。當使用 NFS 掛載的文件系統，已知在一些 Linux 系統上和 SunOS 上出現問題。

- Linux 線性 — 穩定

  唯一發現的問題式 fcntl() 調用，它通過使用 mysqld 的 `--skip-locking` 選項解決。一些人已經報告了 0.5 版中的鎖定問題。

- Solaris 2.5 + pthreads — 穩定

  我們在我們的開發工作中使用。

- MIT-pthreads (其他系統) — 穩定

  自從 3.20.15 版以來，沒有報告的錯誤，而且從 3.20.16 開始沒有已知的錯誤。在一些系統上，在一些操作是相當慢時 (在每查詢之間有 1/20 秒的睡眠) 有一個 “功能失效 ”。當然， MIT-pthreads 可能使任何事情慢一點，但是根據索引的 SELECT 語句通常在一個時幀內完成，因此不應該有一個 mutex 鎖定 /線性的把戲。

- 其他線性實現 — Beta-Gamma

  移植到其他系統仍然是很新的並且可能有錯誤，可能是在 MySQL 中，但是最通常的是線性實現本身。

- LOAD DATA ... ,INSERT ... SELECT — 穩定

  一些人已經認爲他們在這裡發現了錯誤，但是這些經常多是誤解。請在報告問題前檢查手冊！

- ALTER TABLE — 穩定

  在 3.22.12 中有小的改變。

- DBD — 穩定

  現在由 Jochen Wiedmann 維護了。

- mysqlaccess — 穩定

  由 Yves Carlier 編寫並維護。

- GRANT — Gamma

  MySQL 3.22.12.做了很大改變。

- MyODBC（使用 ODBC SDK 2.5） — Gamma

  它與一些程序似乎工作得很好。

- 複製 (Replication) — Beta-Gamma
- BDB Tables — Beta

  主要用於 Transaction 之用途
- InnoDB Tables — Beta
- 自動回護 MyISAM tables — Beta
- MERGE tables — Beta-Gamma
- FULLTEXT — Beta

  全文檢索功能

MySQL 之主要應用領域

由於 MySQL 在 Select/Insert/Update 等之基本 SQL 指令之執行速度非常快速，因此非常適合網站存取紀錄、網站管理、網站資料、網站客户資料之儲存等。因此常見之應用有如下之組合：

1. 線上合約及工作尋找網站 (MySQL + PHP + Linux)
2. 資料應用系統 JDBC + MySQL
3. 網頁系統 (PHP, CGI)
4. 客户管理系統
5. BLOB 資料庫

5.1.2 PostgreSQL

什麼是 PostgreSQL

PostgreSQL 是以加州大學柏克萊分校開發的 POSTGRES 版本 4.2 爲基礎的物件關連型資料庫管理系統（ ORDBMS）。當初由 Michael Stonebraker 教授領導的 POSTGRES 計畫， 是由國防研究計畫局 （ Defense Advanced Research Projects Agency）（ DARPA），陸軍研究處 （ Army Research Office （ ARO）， 國家科學基金會 （ National Science Foundation）（ NSF），和 ESL 公司贊助進行的。 PostgreSQL 繼承最初的柏克萊的程式的一個開放程式碼，它提供了 SQL92/SQL99 語言的支援以及一些其他現代特性。

POSTGRES 所領先的許多物件關連的概念現在已經應用在一些商業資料庫，傳統的關連型資料庫管理系統（ RDBMS）支援一個由命名關連（表）的集合（包括特定類型的屬性 /字段）組成的資料模型。在現代的商用系統中，可能的類型通常包括浮點數、整數、字符串、金額及日期。現在，人們普遍認爲這個模型已經不能滿足未來的資料處理應用的需要了。

POSTGRES 通過一種讓使用者可以很容易擴展系統的方法整合了下面的基本概念，使其能提供可觀的延伸功能：

- 繼承（ inheritances）
- 資料類型（ data types）
- 函數（ functions）

其它特性則提供了附加的功能和靈活性：

- 約束 (constraints)

- 觸發器 (triggers)
- 規則 (rules)
- 事務完整 (transaction integrity)

這些特性將 Postgres 歸類於物件關連 (object-relational) 式資料庫的範疇。這是和那些所謂的物件導向 (object-oriented)，物件導向型資料庫通常並不適合支援傳統的關連型資料庫語言。所以，儘管 Postgres 有一些物件導向的特性，它仍然屬於關連型資料庫的範疇，事實上，一些商用資料庫最近已經引進一些 Postgres 所領先的特性。

PostgreSQL 的歷史

現在被稱爲 PostgreSQL 的物件 -關連型資料庫管理系統（有一段時間被稱爲 Postgres95）是從柏克萊寫的 Postgres 軟體包發展而來的。經過十幾年的發展， PostgreSQL 是世界上可以獲得的最先進的開放程式碼的資料庫系統，它提供了多版本並行控制，支援幾乎所有 SQL 語法（包括子查詢，事務和使用者定義類型和函數），並且可以獲得非常廣泛的（開發）語言支援（包括 C， C++， Java， perl， tcl，和 python）。

柏克萊的 Postgres 計畫 Postgres DBMS 的實現始于 1986 年， 該系統最初的概念詳見 The Design of Postgres。最早的資料模型定義見 The Postgres Data Model。當時的規則系統設計在 The Design of the Postgres Rules System 裡描述。存儲管理器的理論基礎和體系結構在 The Postgres Storage System 裡有詳細描述。

從那以後， Postgres 經歷了幾次主要的版本更新。 第一個 “演示性” 系統在 1987 年便可使用了，並且在 1988 年的 ACM-SIGMOD 大會上展

出。我們在 1989 年發布了版本 1（在 The Implementation of Postgres 裡有描述）給一些外部的使用者使用。爲了回應使用者對第一個規則系統的批評，（ A Commentary on the Postgres Rules System），我們重新設計了規則系統（ On Rules, Procedures, Caching and Views in Database Systems）並在 1990 年 6 月發布了使用新規則系統的版本 2。版本 3 在 1991 年出現，增加了多存儲管理器的支援，並且改進了查詢執行器，重新編寫了重寫規則系統。從那以後，直到 Postgres95 發布前，工作都集中在移植性和可靠性上。

Postgres 已經在許多研究或實際的應用中得到了應用。這些應用包括：一個財務資料分析系統，一個噴氣引擎性能監控軟體包，一個小行星跟蹤資料庫，一個醫療信息資料庫和一些地理信息系統。Postgres 還被許多大學用于教學用途。最後，Illustra Information Technologies（後來並入 Informix）（譯注：現被 IBM 收購。）拿到程式並使之商業化。在 1992 年末 Postgres 成爲 Sequoia 2000 科學計算計畫的主要資料伺服器。

到了 1993 年，外部使用者的數量幾乎翻番。隨著使用者的增加。用于源程式維護的時間日益增加佔用了太多本應該用于資料庫研究的時間，于是該計畫在版本 4.2 時正式終止。

Postgres95 在 1994 年，Andrew Yu 和 Jolly Chen 向 Postgres 中增加了 SQL 語言的解釋器。並隨後將 Postgres95 源程式發布到互聯網上供大家使用，成爲一個開放程式碼的，原先柏克萊 Postgres 程式的繼承者。

Postgres95 所有源程式都是完全的 ANSI C，而且程式量減少了 25%。並且有許多內部修改以利于提高性能和程式的維護性。Postgres95 v1.0.x 在進行 Wisconsin Benchmark 測試時大概比 Postgres v4.2 快 30-50%。除了修正了一些錯誤，其他的一些主要改進還有：

原來的查詢語言 Postquel 被 SQL 取代（在 server 端實現）。在 PostgreSQL 之前還不支援子查詢（但這個功能可以在 Postgres95 裡面由使用者定義的 SQL 函數實現）。重新實現了聚集。同時還增加了對 GROUP BY 查詢子句的支援。 C 程序仍可以使用 libpq 介面函數。

在監控程序方面，新增加了利用 GNU readline 進行交互 SQL 查詢的監控程序（ psql）。增加了新的前端庫， libpgtcl，用以支援以 Tcl 爲基礎的用户端端。一個樣本 shell， pgtclsh，提供了新的 Tcl 命令用于 tcl 程序和 Postgres95 後端之間的交互。徹底重寫了大物件的介面。保留了將大物件倒轉（ Inversion）作爲存儲大物件的唯一機制。去掉了記錄級（ instance-level）的規則系統。但我們仍然可以通過重寫規則使用規則。

PostgreSQL 到了 1996 年， 我們很明顯的看出 "Postgres95" 這個名字已經不能經得起時間的考驗了。于是我們起了一個新名字 PostgreSQL 用于反映最初的 Postgres 和最新的使用 SQL 的版本之間的關連。同時版本號也重新從 6.0 開始，將版本號放回到最初的由 Postgres 計畫開始的順序中。

Postgres95 版本的開發重點放在標明和理解現有的後端程式的問題上。 PostgreSQL 開發重點轉到了一些有爭議的特性和功能上面，當然各個方面的工作同時都在進行。

PostgreSQL 之主要功能

- 支援 Multi-byte (包括中文 Big5)
- 支援 SQL 92/SQL99
- 提供可擴充性的 SQL 機制 (Extended SQL)

- 表級鎖被多版本並行控制取代，這樣就允許讀操作在寫操作活躍時繼續讀取一致的資料，並且令資料庫等待執行查詢時用 pg_dump 進行熱備份成爲可能。

- 實現了許多重要的後端特性，包括子查詢，缺省（值），約束和觸發器等。

增加了附加的 SQL92-兼容的語言特性，包括主鍵，引號標識符，文本字串類型轉換，類型轉換，以及二進制和十六進制整數的輸入。

改進了內建的資料類型，包括新的大範圍日期 /時間類型和附加幾何資料類型的支援。

總體上，後端程式的速度提高了大約 20-40%，而且自 v6.0 起後端的啓動時間縮短了 80%。

PostgreSQL 之主要應用領域

由於 PostgreSQL 具備 transaction 之處理能力 , 對於較 data-critical 之應用系統可以考慮使用。對於 web-based 的應用領域則較不具優勢。

Feature	MySQL
Subselects	4.1
Foreign keys	4.0 and 4.1
Views.	4.2
Stored procedures in multiple languages	4.1
Extensible type system.	Not planed
Unions	4.0.
Full join.	4.0 or 4.1.
Triggers.	4.1
Constrainst	4.1
Cursors	4.1 or 4.2
Extensible index types like R-trees	R-trees are planned to 4.2
Inherited tables	Not planned

postgres delivers each page much more slowly than MySQL.

large-text-area support in MySQL. You may need both, in which case you have to wait for future stable releases of both databases.

slowly adding features like subselects, Postgres is making headway in the performance and stability departments.

and you will make any use of them. If you don't need them or won't use them, then you're probably better off with MySQL and its superior performance.

PostgreSQL 與 MySQL 之比較表

比較項目	PostgreSQL	MySQL
SQL 標準支援	Postgres understands a good subset of SQL92; moreover it understands a good superset of this subset - Postgres adds some object-oriented features to this subset. The team is working on implementing outer joins.	MySQL understands a subset of SQL92, lacking some important features - subselects, for example.
速度	Postgres is somewhat slow but has a lot of options for improving - from -F option - to rewriting a query in a Postgres-friendly way - to use different optimizers hints ("SET KSQO", "SET GEQO"). Postgres forks on every incoming connection - and the forking process and backend setup is a bit slow.	MySQL is very fast on simple SELECTs, but degrades on even a little more complex queries. MySQL handles connections very fast, thus making it suitable to use MySQL for Web - if you have hundreds of CGIs connecting/disconnecting all the time you'd like to avoid long startup procedures.
穩定度	Postgres is much worse in this aspect. Random disconnects, core dumps and memory leaks are usual. Certainly, 6.5 was MUCH better that 6.4, and I hope 7.0 will be better that 6.5, but... To handle more simultaneous connections you need to recompile Postgres with special option - a max number of backends. Tom Lane said: "As of 6.5 no recompile is needed to go up to 1024 backends. If you have a platform hefty enough to support more than that, then you do need a recompile to raise MAXBACKENDS. But the compiled-in limit is normally much larger than the runtime limit being enforced by the postmaster."	MySQL does very good job even on busiest of my sites; it certainly has some problems handling hundreds of connections per second, but these problems are resolvable - MySQL has a lot of tweaking options. Random disconnects and core dumps are exceptionally rare; I have never restored data from backups.

資料一致性	Postgres has transactions/rollbacks; the team is in process of implementing FOREIGN KEYs and all that.	MySQL has some basic provisions for transactions and rollbacks.
伺服器端功能	Postgres has rules, triggers, server-side functions that can be written in C, pgsql, python, perl and tcl languages.	MySQL has simple (and probably inconvenient) mechanism for server-side shared libraries with C functions.
安全性	Postgres has similar features, but a little less fine-grained. For example, if user can connect to a database, (s)he can CREATE TABLE, thus running Denial-of-Service. On the other hand Postgres can limit logins based on different criteria - network segment, ident string, etc.	MySQL has exceptionally good fine-grained access control. You can GRANT and REVOKE whatever rights you want, based on user name, table name and client host name.
大型物件處理	In Postgres, Large Objects are very special beasties. You need to create them using lo_create function and store the result of the function - OID - in a regular table. Later you can manipulate the LOB using the OID and other functions - lo_read/lo_write, etc. Large object support is broken in Postgres - pg_dump cannot dump LOBs; you need to develop your own backup mechanism. Tthe team is working on implementing large rows; this will replace current LOB support.	In MySQL, text and binary LOBs are just fields in the table. Nothing special - just INSERT, UPDATE, SELECT and DELETE it the way you like. There are some limitations on indexing and applying functions to these fields.

變更資料表格	Postgres supports ALTER TABLE to some extent. You can ADD COLUMN, RENAME COLUMN and RENAME TABLE.	MySQL has all options in ALTER TABLE - you can ADD column, DROP it, RENAME or CHANGE its type on the fly - very good feature for busy servers, when you don't want to lock the entire database to dump it, change definition and reload it back.
中文	支援	支援
資料鎖定及同時性	PostgreSQL has a mechanism called MVCC (MultiVersion Concurrency Control), comparable or superior to best commercial databases. It can do row-level locking, can lock rows for writing in one session but give these rows unaffected in another session, etc.	MySQL can do only table locking for reading or writing.
區域語言支援	Postgres compiled with –enable-locale does some job based on its locale settings, and can change locale settings per client (not per database), what is a bit more flexible. Compiled –with-mb (Multibyte support) Postgres can translate on-the-fly between many predefined character sets.	MySQL does some job based on its locale settings, but not much.

5.1.3 其他免費資料庫

Interbase[75] 和 SAPDB[76] 。

5.2 檔案系統 (File System)

作者：謝東翰、李健秋

對於電腦來說，所有的資料都是以檔案 (File) 的形式存在。這些檔案大多是儲存在儲存設備上 (如硬碟、光碟、軟碟、 Zip 碟 .... 等等)，而檔案格式依其用途與資料內容不同而有所不同。由於電腦的儲存設備裡，動輒有上萬個檔案是稀鬆平常的事情。這成千上萬個檔案，就如同圖書館裡有上萬本書，如果不分類整理，將會很難收藏、保存、管理與使用。爲了達到有效率且可靠的檔案存取並加以分類儲存，作業系統由下而上必須同時具備”儲存設備驅動程式”、 ”檔案系統”、與”適當的目錄結構”，三者互相配合，缺一不可。

在 UNIX 的世界中 (包括 GNU/Linux) ，儘管擁有各種類形的檔案，其格式各異，但以系統核心 (kernel) 的角度來看，所有的檔案都是一連串位元組的組合，最後再以一個檔案結束字元 (EOF) 做結束，並無特定的檔案內部格式可言。只有在系統核心之上的函式庫層或應用程式層中，才會去解析、使用檔案內部的格式。事實上， UNIX 作業系統本身對「檔案」的定義是相當廣泛的，它不僅僅只是躺在儲存設備上，以一連串位元組所組成的資料而已，甚至連儲存設備本身、其他系統週邊、網路、系統資源使用 .... 等，都可以當作「檔案」來操作，因爲它們都有相同 (或極相似) 的操作模式，例如：開啓 (檔案、設備、資源)、關閉 (檔案、設備、資源)、讀取 (檔案、設備、資源)、寫入 (檔案、設備、資源)、移動讀寫位置、權限管理 .... 等等。

以下，我們就概括性地介紹 UNIX 作業系統中的檔案運作模式，並以 GNU/Linux 爲中心，介紹其檔案系統的特色。

儲存設備的規劃

這裡所指的儲存設備可分爲兩種，一爲「實體的儲存設備」，二爲「邏輯上的儲存設備」。前者即是我們常見的硬碟、光碟、軟碟 .... 等各式儲存媒體，系統核心擁有適當的驅動程式，可以直接操作這些設備，使得其上的檔案系統層可以利用這些設備來存取檔案資料。通常這些儲存媒體還可以依需要劃分成一個到數個不等的分割區 (partition)，然後分別在其上建置檔案系統，例如我們就經常將一顆硬碟分割成幾塊後來安裝多個作業系統，至於光碟、軟碟 .... 等，由於儲存媒體本身的特性或容量的限制，通常只會有一個分割區 (或完全不分割)。

而「邏輯上的儲存設備」則是介於實體的設備與檔案系統間，它可以將數個實體設備合在一起，看成是一個單一個邏輯設備。檔案系統層在使用此邏輯設備時，就好像在用實體設備一樣，不會有任何的差異性。而採用邏輯設備最大的好處在於可以很彈性地運用系統的各實體設備，以發揮最大的功效或達到特殊的需求。故在很多大型的伺服系統中，此功能的支援是很有幫助的。

在 Linux 核心 2.4 版中，支援了兩種邏輯儲存設備，分別簡介如下：

1. 磁碟陣列 (RAID)

   顧名思義，此邏輯設備可將數顆硬碟組合起來看成一顆單一硬碟來使用。此功能早在 Linux 核心 2.2 版中就已出現了，但核心 2.4 版比起過去有很大的改進，故若要使用磁碟陣列可以考慮直接使用核心 2.4 版。

   一般的磁碟陣列必須要有特別的硬體控制卡才能運作，但除此之外 Linux 核心還擁有軟體模擬磁碟陣列的能力，也就是說機器本身不需要額外添購特別的硬體設備，就能直接將數顆硬碟模擬成磁碟陣列，其穩定性與性能表現都相當優異，而且還有額外的好處：使用

上更有彈性。因爲一般的硬體磁碟陣列多半只能將整顆硬碟加入陣列中，但軟體模擬則可以只使用硬碟上某個分割區，而不影響其他的分割區。

磁碟陣列依用途不同還細分成幾個種類，這些在 Linux 核心中都有支援 :

a) RAID0 Linear: 資料是依序寫入陣列的每個硬碟裡的，寫滿了一顆硬碟後才會寫下一顆。

b) RAID0: 資料是平行分散寫入陣列的各硬碟中的，故在存取單一大檔案時理論上速度會比單一硬碟快上數倍，看陣列中有幾顆硬碟而定。一般而言，軟體模擬的陣列由於受限於硬體本身的設計，使得速度往往不及硬體陣列來得快。但由於此模式中檔案是分散在各硬碟中的，萬一其中有一顆硬碟損壞時，將導至整個陣列的資料流失。

c) RAID1: 通常是以兩顆硬碟組成，其運作原理與 RAID0 相反，資料寫入時會在兩顆硬碟中各寫一份，使得這兩顆硬碟彼此間互爲鏡像，大大提高資料保存的可靠度。然而缺點是寫入的速度慢。

d) RAID4: 通常是以三顆以上的硬碟組成，其中一顆用以保存陣列資料的檢查資訊 (parity information)，而其他硬碟則以 RAID0 的方式平行分散讀寫。當其中一顆 (非保存檢查資訊的那顆) 硬碟毀損時，更換新硬碟後系統會自動以檢查資訊重建整個陣列資料，故資料不會流失。

e) RAID5: 通常是以三顆以上的硬碟組成，運作原理與 RAID4 幾乎相同，只有一點不同，就是檢查資訊是分散在各硬碟中，而非集中在其中一顆，如此不會因保存檢查資訊的硬碟損毀導至資料無法挽救。這也是最常見的模式。

在 GNU/Linux 中，使用磁碟陣列時只能將整個陣列看成單一一個分割區，我們不能在陣列裡頭劃分多個分割區來使用。

2. LVM (Logical Volumn Manager):

LVM 與磁碟陣列用途不同，它雖然也是將數個分割區或儲存設備合起來看成一個單一的磁碟來使用，但其目標並不是在提升資料存取的速度與資料保留的可靠度，而是賦與系統執行時期 (run time) 可以動態調整儲存空間大小的能力。

在實體的儲存設備中，一旦分割區劃分好之後，其容量大小就固定了，如果將來容量不夠了必須重新劃分時，我們必須先將原分割區，或甚至整顆硬碟的資料都備份出來，因爲重新分割會毀掉其上所有的資料。

然而 LVM 卻不需如此，它是將數個分割區與儲存設備合起來組成一個 Volumn Group, 而這個 Volumn Group 內可以再劃分割區，分別建置檔案系統。萬一將來某分割區不夠用時，我們可以直接調整各分割區的大小，從別的分割區挪一些空間過來用，或甚至如果有新的儲存設備連接上來時，以可以將它直接加入此 Volumn Group 中 以擴充空間。而這些調整過程可以在系統執行時期進行，不需要先備份原有的資料。

當然，使用 LVM 必須要有適當的檔案系統配合，才能完全發揮其功能，因爲一般檔案系統的容量是在建置時就固定了，無法在執行時期任意改變，故就算 LVM 可以任意調整其分割區的大小，若檔案系統本身無法跟著調整結果還是枉然。目前這部分還在持續發展中，但已有不錯的成果了。

快取與直接存取

一般而言，當我們要從儲存設備中存取資料時，都是透過檔案系統的操作來完成。這些資料檔案都是以「資料區塊 (block)」爲單位保存在檔案系統中，而系統在存取時也是以整個資料區塊爲單位來讀寫。資料區塊的大小依作業系統、機器硬體結構、以及儲設備容量大小等不同而有不同，以 GNU/Linux 系統爲例，常見的資料區塊大小爲 1024、 2048、或 4096 個位元組。

在儲存設備與檔案資料讀取介面之間，還有一層稱之爲「資料區塊快取層」，由檔案系統直接管理，其主要的功能是做儲存設備上的資料區塊快取。當系統要讀取某些區塊時，檔案系統在眞正去讀取儲存設備之前，會先檢視快取層的資料，如果找到所需的區塊就直接傳回，若找不到時檔案系統才會眞正去讀取儲存設備，並將讀取的結果暫時保存在快取層中。同理，如果系統要寫入某些資料區塊時，檔案系統也會先將要寫入的內容保存在快取層中，等系統較不忙碌或儲存設備沒在使用時才做眞正的寫入動作。因此，透過快取層的運作，可以減少直接由儲存設備存取資料的機會，因而大幅提升系統的讀寫效率。

然而快取層的優點同時也帶來的負面效應。由於新寫入的資料不是立即寫入儲存設備中，故如果在此過程中突然發生了硬體錯誤或斷電，將會造成資料的流失。同時，在某些系統中，其快取層的運作效率可能也不盡理想。因此，在某些特殊的應用場合中，如保存極重要的資料，或大型資料庫系統的管理運作等，我們需要直接讀寫儲存設備，而不需要經由檔案系統爲媒介。爲了達到此需求，在一般的 UNIX 系統中就提供了稱爲 Char-RAW Device 的存取介面以供使用。

在過去， Linux 核心中並沒有類似的介面，主要是因爲 Linux 核心的檔案系統快取層運作極有效率，足以應付各種特殊需求，而讓應用程

式直接存取儲存設備的資料不見得會做得更好，故不需要加入類似一般 UNIX 系統中常見的 Char-RAW Device。然而此情況在 Linux 核心 2.4 版中已有改變了，在這一版中開始有了 Char-RAW Device 的支援，以因應特殊的應用需求。然而它的使用方式與一般 UNIX 上的稍有不同。在一般 UNIX 系統中所有的儲存設備都有一個相對應的 Char-RAW Device, 因此當系統的內接的儲存設備一多時，其相對應的 Char-RAW Device 也就會有一大堆，然而這種可直接存取的設備又不常用，故這可能造成管理上的麻煩。而在 Linux 核心的做法中，在平常時候並沒有任何 Char-RAW Device 存在，當應用程式需要使用時，才經由工具程式將需使用的儲存設備 (或分割區) 與 Char-RAW Device 連繫，因而賦與它可被直接存取的能力。

各種檔案系統

檔案系統是建置於儲存設備的分割區之上，用以儲存檔案資料。一個檔案系統在使用之前必須先「掛上 (mount)」，將它掛在系統樹狀目錄結構的某個點上。同理，當某個分割區的檔案系統不使用時，可以將它「卸下 (umount)」，如此系統會將尚未寫入的資料寫回去，並將它從系統的樹狀目錄中除去。

所謂「樹狀目錄結構」意指系統檔案放置的位置整個看起來就像一棵倒過來的樹一樣，由最上層的「根目錄」開始，其下可以有檔案，也可以有目錄 (即微軟系統中俗稱之「資料夾」)，而各目錄下還可以有子目錄與檔案，如此一路繁衍，形成狀似枝葉茂密的大樹。而每個檔案在此目錄樹的位置，就稱之爲「路徑 (path)」。此樹狀目錄結構的檔案配置方式是 UNIX 的一大創舉，之後影響深遠，有許多作業系統都採取這樣的設計方式，包括微軟的作業系統。而與微軟作業系統不同的是，在微軟系統中每個儲存設備或分割區都以一個英文字母來做代表，例如 A:

是軟碟， C: 是硬碟的第一個分割區，而光碟可能是 D: 等等。在 UNIX 的世界中並不採用這樣的方式，如前所述，所有建置於儲存設備或分割區上的檔案系統都是以「掛上」的方式連上整個目錄樹，因此，目錄樹上任何一個「子目錄」都可以是一個掛入點，而此後系統在做資料存取時，就與系統其他的目錄檔案存取方式一致。

由於使用前「掛上」與使用後「卸下」的動作在微軟的作業系統中都沒有，可能造成已用慣微軟系統的朋友的不適應。故在 Linux 核心中還支援「自動掛上」與「自動卸下」 (automount) 的功能，在經過適當的設定後，所有的掛上與卸下的動作都可以自動進行，使用上就與微軟的系統一般無異。

在 Linux 核心中支援各式各樣的檔案系統，使得 GNU/Linux 在各種平台間的整合能力極佳。而這些爲數眾多的檔案系統，全部由「虛擬檔案系統 (Virtual File System)」管理，它提供了統一的操作介面供系統使用，故系統不需要因操作的檔案系統不同而改變其操作方式。在這些檔案系統中，屬於 UNIX 類的多半都有完整的功能實作，例如檔案屬性、檔案所有人與使用權管理、以及使用者可使用的容量管制 (Quota) 等等。而不屬於 UNIX 類 (如微軟平台的) 可能就會欠缺某些功能，這是因爲該檔案系統原本設計上的限制。

Linux 核心所支援的檔案系統大至分類如下 :

1. 一般檔案系統 :

   這些檔案系統很多是在早期的 Linux 核心版本中就有支援的，這之中包含了許多其他作業系統平台的檔案系統，包括 :

   a) ext2:

      Ext2 是 GNU/Linux 系統中標準的檔案系統，其特點爲存取檔案的效能極好，對於中小型的檔案尤佳，這主要得利於其資

料區塊快取層的優良設計。其單一檔案大小與檔案系統本身的容量上限與檔案系統本身的資料區塊大小有關，在一般常見的 x86 電腦系統中，資料區塊最大爲 4KB, 則單一檔案大小上限爲 2048GB, 而檔案系統的容量上限爲 16384GB。但由於目前核心 2.4 所能使用的單一分割區最大只有 2048GB，因此實際上能使用的檔案系統容量最多也只有 2048GB。

b) BSD 平台檔案系統 :

BSD 平台是另一類的自由 UNIX 作業系統，包括 FreeBSD、NetBSD、 OpenBSD 等，其中 FreeBSD 最爲常見。其所採用的檔案系統稱之爲 ufs。除了各種 BSD 平台以外，此類的檔案系統也用於 SunOS、 NextStep 等系統中，但格式稍有不同。以 FreeBSD 的爲例，其 ufs 檔案系統最大的特色是可以在分割區上在分割子區塊，稱之爲 Slice，每個 Slice 可分別用來建置檔案系統，其使用方式就和我們平常在使用分割區一樣。此特色使得我們只需準備單一一個磁碟分割區，就可以安裝一個完整的 FreeBSD 系統。

c) 微軟平台檔案系統 :

微軟平台的檔案系統包括 msdos (使用於 MS-DOS), vfat (使用於 Win95, Win98, WinMe 等), 與 ntfs (使用於 WinNT 等)。其中 vfat 與 msdos 相當類似，差別只在於 vfat 支援長檔名而已。除此之外， Linux 核心還支援 umsdos 檔案系統，它可以直接在 msdos 檔案系統中規劃一塊區域來安裝 GNU/Linux 作業系統，而無須額外劃分割區，這在早期很常見，特別是當電腦的硬碟很小，已安裝了 MS-DOS，而無法再重新分割其他分割區時。然而，使用 umsdos 其檔案讀寫效率不好，故不建議將 GNU/Linux 作業系統安裝在其上，最好還是使用 ext2。

msdos 與 vfat 很早以前就已支援完整的讀寫功能，但 ntfs 截至目前爲止還只有讀的功能而已，寫的功能仍在發展中。值得一提的是 vfat 中的 codepage 支援，它可以讓 vfat 長檔名中使用各國的文字，包括我們台灣常用的 Big5 (其爲 cp950)。

d) 光碟媒體檔案系統 :

此類檔案系統包括 iso9660 以及最近新開發的 udf, 前者用於一般的 CDROM，後者用於 DVD 片，二者都是唯讀檔案系統。其中 iso9660 還支援了微軟的 Joliet 的延伸規格，它可以讓 CDROM 的長檔名可以內含 Unicode 格式。

e) 其他平台檔案系統 :

以下的檔案系統在一般情況下可能較不常用，茲簡述如下 :

i. minix: 此作業系統是由阿姆斯特丹的 Andrew S. Tanenbaum 教授爲教學用所發展的類 UNIX 作業系統。由於是爲了教學用，故其功能較爲簡單陽春。而 Linux 核心在最早期草創階段，就是在此作業系統上的。

ii. hpfs: OS/2 平台的檔案系統。

iii. hfs: 麥金塔電腦的檔案系統。

iv. adfs: Acorn Disc 檔案系統，用於 ARM Risc PC 的 RiscOS 作業系統上。

v. affs: 用於 Amiga 的 AmigaOS 作業系統上。

vi. bfs: 用於 SCO UnixWare 作業系統上，在開機時載入核心檔案用。

vii. efs: 用於舊的 SGI IRIX 作業系統上。

viii. vxfs: 此爲 Veritas VxFS 檔案系統，用於 SCO UnixWare，也見於 Solaris、 HP-UX 及其他作業系統上。

ix. qnx4fs: 用於 QNX 4 作業系統。

x. sysv: 用於 SCO, Xenix 及其他商業版 UNIX 作業系統。

2. 日誌式檔案系統：

在之前已提過，由於檔案系統都有快取層參與運作，如不使用時必須將檔案系統卸下，以便將快取層的資料寫回磁碟中。因此每當系統要關機時，必須將其所有的檔案系統全部卸下後才能進行關機。

如果在檔案系統尚未卸下前就關機 (如停電) 時，下次重開機後會造成檔案系統的資料不一致，故這時必須做檔案系統的重整工作，將不一致與錯誤的地方修復。然而，此一重整的工作是相當耗時的，特別是容量大的檔案系統，而且也不能百分之百保證所有的資料都不會流失。故這在大型的伺服器上可能會造成問題。

爲了克服此問題，業界經長久的開發，而完成了所謂「日誌式檔案系統 (Journal File System)」。此類檔案系統最大的特色是，它會將整個磁碟的寫入動作完整記錄在磁碟的某個區域上，以便有需要時可以回朔追蹤。由於資料的寫入動作包含許多的細節，像是改變檔案標頭資料、搜尋磁碟可寫入空間、一個個寫入資料區段等等，每一個細節進行到一半若被中斷，就會造成檔案系統的不一致，因而需要重整。然而，在日誌式檔案系統中，由於詳細紀錄了每個細節，故當在某個過程中被中斷時，系統可以根據這些記錄直接回朔並重整被中斷的部分，而不必花時間去檢查其他的部分，故重整的工作速度相當快，幾乎不需要花時間。

目前在 GNU/Linux 上的日誌式檔案系統，除了 ext3 之外，其餘均爲來自業界的貢獻，茲簡介如後：

a) ext3: 顧名思義，它就是 ext2 的下一代，也就是在保有目前 ext2 的格式之下再加上日誌功能。目前它離實用階段還有一段距離，也許在下一版的核心就可以上路了。

b) reiserfs: 此檔案系統爲 Threshold Networks、 Emusic.com、 Bigstorage.com 等公司所支持開發，它目前已完整地與核心 2.4 整合在一起，且發展已接近成熟。其特色除了日誌功能以外，其在數量龐大的小檔案存取效率相當驚人，在某些情況下可以達到 ext2 檔案系統的三至四倍以上。

c) xfs: 此檔案系統爲 SGI 公司所開發，已移植到 GNU/Linux 系統上，但尚未整合到 Linux 核心中。除了日誌功能以外，其最大的特色是高延展性，可以有效率地處理超大型檔案。

d) jfs: 此檔案系統爲 IBM 公司所開發，已移植到 GNU/Linux 系統上，但尚未整合到 Linux 核心中。其主要特色與 xfs 相似。

這些新開發的檔案系統如 reiserfs、 xfs、及 jfs 除了日誌功能較傳統的檔案系統 (如 ext2、 ufs .... 等) 較爲進步外，通常還擁有傳統檔案系統難以達到的能力，例如高處理效率、高延展性等等，這些額外的特色主要來自檔案系統內部的資料結構與演算法的改進。這類檔案系統都使用了平衡樹 (balanced tree) 來管理檔案資料區塊以及硬碟未使用的區塊，使得在數量龐大的運作環境下可以顯著提升效率，故相當適於大型資料庫與資料搜尋的應用場合上。然而 ext3 在設計時就沒有使用這些先進的設計，主要是爲了與目前的 ext2 相容之故。

3. 網路檔案系統 :

網路檔案系統並非建置在機器本上的儲存設備上，而是用以將遠端機器的檔案系統經由網路連線掛到本身的機器上，使其使用上就與一般的檔案系統無異。 Linux 核心支援了數種網路檔案系統，包括 :

a) NFS: NFS 是此類檔案系統的代表，它是由 Sun Microsystems

公司設計發表的，現已成爲各 UNIX 系統的標準配備之一。

b) Code (Coda?) 檔案系統：此檔案系統與 NFS 類似，但擁有較 NFS 還先進的特色，例如斷線操作、安全性認證等。

c) SMB: 此檔案系統可以將微軟作業系統的「網路芳鄰」分享出來的磁碟或檔案系統掛上來使用。

d) NCP: 此檔案系統可以將 Novell NetWare 的 IPX 分享磁碟掛上來使用。

4. 虛擬檔案系統：

在 Linux 核心中，有許多檔案系統並不是用來存取實際的檔案資料。這些檔案系統在掛上後，我們可以見到其內部的檔案目錄，但實際上這些檔案目錄都不佔任何儲存設備的儲存空間，因爲它們根本不是來自任何儲存設備，它們都是系統核心所製造出來的檔案「影像」。

這麼做的目的主要有兩個，其一就是用以方便存取系統內部資訊。前面我們已說過， UNIX 世界對「檔案」的定義相當廣泛，只要它們的操作方式與一般檔案一樣或相近即可。因此，就利用這樣概念，將一些系統資訊以檔案的方式出現在檔案系統中，讓應用程式可以隨時存取使用。由於它們的內容來自核心內部的資料鏡像，故只要核心狀態改變，它們的內容也會立即改變。

另一個目的爲有效率配置系統資源，供應用程式使用。由於應用程式要使用系統資源時，傳統的 UNIX 的做法都是去操作系統的設備檔 (device files)，因此系統各裝置的設備檔必須存在，其對應的裝置才能驅動。然而，很多時候我們無法預知系統會有那些裝置存在，在大型的系統中會有相當多的裝置，但小型的桌上系統則否。如果要考慮所有可能的情況而將所有的系統設備檔都準備好，顯然

不是件經濟的做法。因此，此問題就可以考慮由虛擬檔案系統來處理，當系統眞的有該項裝置並且系統核心也偵測到時，系統核心就會自動產生一個虛擬的設備檔案以代表該項裝置，如果該裝置移除了，該設備檔就會消失。

虛擬檔案系統隨著未來的系統裝置越來越複雜、應用越來越多樣化而不斷增加其種類與普及率。目前 Linux 核心中包含了如下的虛擬檔案系統：

a) proc: 用以存取系統核心狀態資訊。此檔案系統同時也出現在許多新設計的 UNIX 系統上，使用相當廣泛。其內含的資訊包括：系統資源分配狀況、機器硬體組 態、各設備目前的狀態、網路系統狀態與可調整選項、以及所有執行中的行程 (process) 狀態等等。

b) devfs: 用以取代傳統的設備檔，使用此檔案系統時所有的系統設備檔都會依一定的規則存在於樹狀目錄群中，而且只有該裝置存在時才會存在。此爲核心 2.4 的新設計，對於大型系統的延展性有相當大的幫助。目前此檔案系統尚在開發階段，在不久的將來將會被廣泛使用。

c) devpts: 用於接受外部網路連線 (telnet) 的虛擬終端機埠。每一個外來的網路連線本機都必須要準備一個虛擬終端機埠來處理它，而每個虛擬終端機埠就是一個設備檔，故在此情況下使用虛擬檔案系統是最佳的解決方式。未來當 devfs 被廣泛採用時，devpts 的功能將完全由 devfs 取代。

5. 特殊用途檔案系統：

此類檔案系統在一般應用上的使用機會不大，多半只會在特殊的場合上才會使用，例如嵌入式系統上。目前有越來越多的廠商以

Linux 核心爲基礎來開發嵌入式裝置，而這類裝置由於所擁有的儲存設備很小 (或甚至完全沒有)，同時主記憶體的容量也相當有限，故往往需要特別設計的檔案系統。

目前 Linux kernel 支援下列的特殊用途檔案系統 :

a) cramfs: 此爲 Compressed ROM File System，爲唯讀檔案系統，其容量上限只有 256MB，用於嵌入式裝置。

b) romfs: 此爲非常小的唯讀檔案系統，用於唯讀的儲存媒體。

c) jffs: 此爲日誌式快閃 (Flash) 檔案系統，用於嵌入式裝置。

d) tmpfs: 此檔案系統可以用來將檔案暫時保存在主記憶體 (RAM) 中，而且其容量可以隨著保存檔案的量而增減。

e) ramfs: 此檔案系統也用於將檔案暫時保存在主記憶體中，與 tmpfs 類似。

參考資料 :

1. Linux Kernel 2.4 Documentation.

2. Software-RAID HOWTO [77]

3. Wonderful World of Linux 2.4 [78]

4. Char-RAW Device [79]

5. Overview of the Virtual File System [80]

6. Design and Implementation of the Second Extended Filesystem [81]

6. Journal File Systems [82]

7. Reiserfs [83]

8. XFS [84]

5.3 編碼 (Encoding)

作者：謝東翰

5.3.1 電腦編碼系統簡介

電腦系統中資料儲存的最基本單位是「位元 (bit)」，而每個「位元」所能代表的數值只有 0 與 1 而已。因此當我們要用電腦做資料的儲存與處理時，我們通常會將連續幾個位元和起來看成一個單位，稱之爲「位元組 (byte)」，其所能代表的數值就不會只限於 0 與 1，而可以達到 2^n-1 (其中 n 爲該位元組中所包含的位元個數)，如此我們就可以將我們想記錄在電腦系統的符號一一編號，以位元組爲單位儲存在電腦中，這就是編碼 (Encoding) 的基本觀念。

由於早期的電腦系統是發源於美國，因此最早的編碼系統也是發源於此。由於他們的資料只需數字、 26 個英文字母 (包括大小寫)、標點與其他特殊符號、外加一些電腦系統的控制碼即可，因此當時一個位元組的大小只需 7 個位元即可包含所有所需的資訊，總共可容納 128 個符號，這也就是大家所熟知的 ASCII 編碼。

然而對於歐洲語系而言， 7 個位元的編碼空間不符合所需，因爲除了 26 個英文字母外，歐洲許多國家還需要拉丁字母、特殊字母的上下標與其他符號等，才能完整表示他們的語言。因此原來的編碼系統必須再擴充，由 7 個位元變爲 8 個位元，以容納那些多出的符號，這就是 ISO8859 的編碼標準。而依地區語系的不同， ISO8859 的編碼標準又分成幾個部分，例如 ISO8859-1 所包含的符號可使用於英語、丹麥語、芬蘭語、德語 .... 等，而 ISO8859-2 則可用於英語、德語、羅馬尼亞語、波蘭語 .... 等。每一個部分的 0-127 碼的編碼方式與 ASCII 都是一樣的，只有在 128-255 碼才依地區不同而不同。如此不同地區的人們可以

選用不同部分的 ISO8859 編碼來處理他們的資料。

然而到了亞洲地區，上述的 8 位元編碼又不夠用了。因爲亞洲地區不再是拼音文字，因此多出了成千上萬個符號，我們已無法仿照當初 ISO8859 的做法，弄出屬於中文的部分，或屬於日文的部分。唯一的作法只有合併幾個位元來共同表示一個符號，如此才有足夠的空間來容納一個地區所有的文字。例如我們台灣地區最常見的 BIG5 編碼，就是將兩個位元組合起來看成一個字。但當一份文件中同時要有中英文時，意味著我們要有一個規則可以區分這個位元組應該視爲 ASCII 碼，或者它與它的下一個位元組應合起來視爲一個 BIG5 碼。爲了定出這樣的規則，就表示說我們無法全部使用整個雙位元組的所有空間 (共 65536 個值)。就如我們的 BIG5 碼，其所包含的字數就只有 13000 多個。

以上就地區語文使用似乎已經足夠了，然而在某些場合下，我們仍需要在同一份文件中記錄多國文字。以 BIG5 編碼爲例，它可以確保中文字與 ASCII 碼可以同時存在，不至於混餚 (事實上對於其他語系而言，他們的編碼方式多半都是如此 : 可以讓該地區的文字與 ASCII 碼同時存在)，但如果我們要同時表示中文、日文、韓文的文字呢？這裡就眞正牽涉到多國語文的編碼範籌了。而編碼系統的擴充就有兩種形式 :

1. 文法式 (Modal) 編碼系統 :

   這裡最有名的就是 ISO2022 編碼。它的特徵是讓一個字串有「狀態」的區別，而不同「狀態」的改變則以幾個特殊位元組值來代表。例如當一個字串中讀到了位元組值 a 與位元組值 b 時，就表示字串進入狀態一，若讀到位元組值 a 與 c 時就表示字串進入狀態二 .... 等。然後我們可以定義狀態一以後的字串應解釋爲日文 JIS 編碼，狀態二以後應解釋爲簡體中文的 GB 編碼 .... 等等。如此便可以有系統地將各國的編碼方式整合進來。

很明顯的，這樣的編碼方式有一個限制，就是可以整合進來的編碼系統不能與狀態改變標記位元 (如上例的 ab, ac .... 等) 衝突，否則的話就無法整合進來。我們的 BIG5 編碼就是因爲衝突了，所以無法整合到 ISO2022 中。而且，這樣的編碼方式很複雜，程式實作困難，因此近年來較少有人使用了。

2. 定長式編碼系統 :

此編碼系統是仿照 ASCII、 ISO8859 的概念，給定一個固定長度的位元數，試圖將世界上所有的符號都包含進來，並依序給它們編號。此方式最著名的就是 Unicode (或稱 UCS2) 與 UCS4。以 Unicode 爲例，其每個「位元組」的長度固定爲 16 個位元，即拿兩個傳統 8 位元的位元組合併成一個新的位元組，然後編號從 0 到 65535 全部拿來使用，其中 0 到 127 是與 ASCII 碼相容的。如此方式，大概可以包含世界上主要的語言與符號如歐美地區、中日韓越、中亞地區 .... 等等。至於 UCS4 則是 Unicode 的再擴充，其每個「位元組」有 32 個位元，其中 0 到 65535 是與 Unicode 相同的，其餘則是用來編入世界其他少數語言與罕用文字。

此編碼方式很明顯的好處就是沒有了文法式編碼系統的限制，同時程式容易實作，只要提供一個適當的轉碼表，我們就可以很容易地將地區常用編碼系統轉換爲 Unicode 或 UCS4。此轉碼表可能相當大，特別是在繁體中文的情況，但對於目前電腦系統的能力 (記憶體容量與運算速度 .... 等) 而言已不是問題。因此，此編碼方式也逐漸成爲世界的主流。儘管在某些地區仍然對此編碼方式有意見，但在可預見的未來它將會越來越普遍。

然而，定長式編碼系統並非完全沒有限制的。最明顯的一點就是目前的電腦硬體與程式語言都是將一個「位元組」定義爲 8 個位元長，而非 16 或 32 個位元長，而且考慮到相容性的問題，在可預

見的未來恐怕不會有全面性的改變。同時由於不同 CPU 對於「位元序」的解釋不同，使得同樣是 16 或 32 位元的一個數值，在不同的 CPU 下會呈現不同的值 (例如 32 位元下在 x86 CPU 所呈現的數值 1，若將同樣的位元資料移到其他 RISC CPU 下可能呈現出 16777216)。因此定長式編碼多半只能用於正執行中的程式內部，以便於資料處理。若程式需要做資料的輸出入時，最理想的方式是將它轉爲適當的「多位元組編碼」(見後述) 然後才進行。

5.3.2 GNU/Linux 下的編碼與字集

在很多情況下，我們經常將「編碼 (Encoding)」與「字集 (Character Set)」混爲一談，但實際上它們各自有明確的定義的。所謂的「字集」是一組符號 (或文字) 的集合，而「編碼」則是將這一組符號 (或文字) 以適當的方式編入位元組中，以便電腦能夠表示與儲存。在許多情況下，由於一個編碼方式所能容納的字數是有限的，也因此就限定了它所能使用的「字集」，故有時我們會將「編碼」與「字集」視爲同一個東西。

在使用 glibc-2.2.X 系列的 GNU/Linux 系統中，由於已完整支援了 I18N 架構，使得世界各地的語系資料庫都已完整地建立在系統底層的函式庫 (libc) 中，這其中也包含了各語系所用的字集與其相對應的編碼系統。以我們中文爲例，兩岸三地可能會常用到的編碼系統包括：

1. BIG5: 台灣地區最常見的編碼方式。目前 glibc 中的 BIG5 是以微軟公司所定義的 CP950 爲基礎，外加文鼎公司所採用的倚天外字集與 Unicode 轉換表而成。

2. EUC-TW: 此編碼即等價於教育部所編定的 ISO10646 國家標準編碼，目前已內含了所有與 UCS4 可以互轉的中文字，其他尚未定義

與 UCS4 可互轉的中文字則要等定義完成後才會納入。

3. BIG5HKSCS: 此爲香港地區所使用的 Big5 與香港政府增補字集。它與 BIG5 編碼是相容的，並加以擴充以容納其他罕用字。

4. GB2312: 此爲大陸地區最通行的編碼方式，可容納常用的簡體字集。

5. GBK: 此爲 GB2312 的擴充，除了容納原本 GB2312 中的簡體字以外，同時還包含了繁體字與其他罕用字。

6. GB18030: 此爲最近新定義出的字集編碼，與 GB2312 相容。

7. Unicode 與 UCS4: 其中 UCS4 在 GNU/Linux 系統中被用作轉碼系統的「基底字集」(見後述)。

8. UTF-8: 此爲 Unicode 的多位元編碼形式，可用於 Unicode 編碼的輸出。

以上的編碼系統在 GNU/Linux 中都有完整的支援，至於教育部新定的 Big5+ 目前則不在支援之列。

5.3.3 GNU/Linux 的轉碼系統

在具備完整的 I18N 環境支援的 UNIX 系統中 (包括 GNU/Linux 系統)，其編 碼系統的轉換有兩種方式 :

1. 一般轉換介面 : 此即爲 libc 中的 iconv() 編碼轉換介面，它可以做到各種編碼的轉換。然而，在一般的 UNIX 系統中，並不見得系統同時支援編碼系統 A 與編碼系統 B, 就一定能做到 A 與 B 之間的互轉，即使 A 與 B 內含相同的 (子) 字集。但在 GNU/Linux 底

下，只要 A 與 B 內含相同的 (子) 字集，則可以直接透過 iconv() 做轉換。例如將 BIG5 轉成 BIG5HKSCS 或反過來，或者 GB2312 與 GBK 與 GB18030 之間的互轉。當然，各字集與 UCS4 之間的互轉都不是問題，原因就是 glibc 是以 UCS4 做爲轉碼系統的「基底字集」。

所謂的「基底字集」意指各地區的語文符號在系統內部的表式方式，而且它還是各地區所有語文符號的集合。根據此定義，表示我們可以將世界各地所有的編碼方式轉換成「基底字集」來統一處理 (理論上)。而 glibc 所採用的「基底字集」的編碼方式就是 UCS4, 它是定長編碼系統，採用 32 位元的位元組來編碼，它是目前已定義最大的一個字集的編碼方式，而且它仍持續擴編世界各地的文字與符號。

在 GNU/Linux 下如果要透過 iconv() 進行轉碼工作時，它實際上是先將該編碼轉成基底字集，然後再轉成目的編碼，如此便二確保只要編碼中有共同的 (子) 字集時，轉換可以順利進行。

便對於沒有共同子字集的情況呢？例如 BIG5 與 GB2312 之間的互轉，在 GNU/Linux 下也提供額外的 iconv() 模組來進行，而這些都統一在 iconv() 介面之下，故在實際使用上不會感到任何差別。然而這類的額外模組目前並未包含所有可能的轉換情況，但仍持續在建構中。例如 BIG5 與 GB2312 的互轉模組已於日前完成，目前尚在實驗版的 glibc-2.3 測試中，預計不久的未來等 glibc-2.3 正式發表，並且大部分的 GNU/Linux 系統都換裝 glibc-2.3 之後，中文繁簡轉換的部分將不是問題。

2. 區域資料庫編碼轉換介面：此轉換介面是與系統區域化資料庫 (locale) 相關的，也是絕大部分的 I18N 程式會使用的介面。由於不同的區域化資料庫所採用的編碼系統是固定的，例如我們台灣地區

的區域化資料庫名稱爲 zh_TW (或 zh_TW.Big5), 其採用的編碼系統即爲 BIG5。而區域化資料庫主要的任務只有在讓同一份程式碼可以處理各地區的語言，因而可以適用於各地區，而不需要額外修改程式本身，故它的任何不在於處理多國語文的問題。因此，這個轉碼介面只負責該資料庫所採用的編碼，與系統基底字集所採用的編碼系統互轉而已。它不包含如上述 iconv() 轉碼介面那樣，可以做到任意編碼系統間的互轉。

此轉碼介面最大的用途在於方便程式做文字處理。因爲地區原本的編碼系統多半是採用「多位元組」編碼方式，例如 BIG5 就是將兩個 8 位元的位元組組合起來使用，故稱「多位元組編碼」。但當我們要在字串中區分中文字與英文字母時就會遇到麻煩，因爲前者是以兩個位元組來代表，後者確是以單一一個位元組來代表，故我們要在程式中額外寫入 BIG5 碼的編碼規則才行，而這樣的程式就不能適用與其他的編碼方式了。因此，當我們用此轉換介面 將它轉成定長編碼方式時，由於每個字 (不論中英文) 都是故定的長度 (即位元組數)，我們稱之爲「寬字元 (wide character)」，這使得我們可以很方便做字串處理。同時，我們的程式中也不必加入 BIG5 的編碼規則，因爲這些都在此轉碼介面中處理掉了，因此，這樣寫出來的程式就是一個支援 I18N 的程式，可以在不需要修改程式碼本身的情況下適用於各語系與編碼系統。

然而，我們在前面也提過，定長編碼方式 (即寬字元) 只能用於運作中的程式內部，如果在做資料輸出入時可能因 CPU 的位元序不同與各硬體系統的限制而出錯。因此，當我們要做資料交換時，最好的方式是將它轉回多位元組編碼，才能確保不會有問題。這在第 2 點的轉換介面是不成問題的，因爲基底字集與其多位元組編碼之間的對應是定義在區域化資料庫中的；但在第 1 點的轉換介面中，

特別是 Unicode 或 UCS4，而且包含了如中日韓等多國語文資料的情況又如何呢？對於 Unicode 而言，其所對應的多位元組編碼爲 UTF-8, 它可以將所有 Unicode 碼轉換成不定長度的位元組，視其 Unicode 碼值而定，而且轉出來的多位元組與 ASCII 編碼相容，故可以適用於所有的 UNIX 應用程式。至於 UCS4 所對應的多位原組編碼則爲 UTF-16, 它是將兩個 UTF-8 字元組合而成的。

參考資料：

1. CJKV: Information Processing, by Ken Lunde, Publish: O'Reilly, ISBN: 1-56592-224-7

2. Documents of glibc-2.2: man pages and info pages.

第六章

開發環境

作者：謝東翰

6.1 總論

有人說，自由軟體是程式開發者的天堂，因爲所有的原始碼都是公開且可以自由使 用的，使得我們可以很容易以既有的程式碼爲基礎，修正程式的錯誤、加強程式的 功能，甚至發展出全新、更強、更好用的程式。

然而，通往這個天堂的道路，除了公開且可自由使用的原始碼以外，豐富且功能強 大的程式開發工具更是不可或缺的因素。舉凡各種語言的編譯器、程式除錯維護工 具、整合開發環境、還有形形色色的直譯式語言，在自由軟體世界中都隨處可見。 在這許許多多的開發工具相互搭配下，不儘可以簡化程式開發的過程，更能協助我 們做後續的維護與管理的工作。因此，善用這些開發工具，是每位程式設計師必備 的基本技巧。

6.2 編譯器家族 GCC

GCC 可以說是 Richard Stallman 所創立的 GNU 計畫中最重要的作品之一，它提供 了自由軟體世界高品質的編譯器 (compiler)，實現了我們在自由軟體平台上開發程 式的夢想。如果沒有 GCC，恐怕今日我們也不會有這麼多形形色色的自由軟體可用。

早期 GCC 的發展方向是以 C/C++ 與 Objective C 等語言爲主，故 “GCC” 的函義 即爲 “GNU C Compiler”。但發展到現在， GCC 的內函已不只是 C 與類似 C 的程式 語言而已了，它同時還包含了許多其他語言的編譯器，例如 GNU Ada Translator (gnat), Java (gcj), Fortran 77 (g77), Modula-2, Chill, Pascal (gpc) .... 等等，有些發展已接近成熟，有些尚在發展中，而有的則因爲沒有人在用，故不再維 護了。除此之外， Cobal 與 Fortran 95 的編譯器則還在早期規劃階段。這些編譯器全部共用同一組程式編譯最佳化的引擎，使用相同的浮點數運算模組，同時也都享有 GCC 高移植性的特色。因此， GCC 事實上包含了一個很大的編譯器家族，而 “GCC” 的函義也隨之轉變爲 “GNU Compiler Collection”。

GCC 一個很大的特色是高度可移植性，目前已知有超過三十種硬體平台與作業系統 可以執行 GCC，其中硬體平台包括了 : x86, ia64, alpha, hppa, m68k, Power PC, mips, IBM rs6000, sparc/sparc64, ... 等，而作業系統則從 Microsoft 平台 (DOS/Win32) 到 IBM OS/2 到各家的 UNIX 都有。此高度可移植性正是 GCC 廣爲流 傳散佈的主要原因。在許多商業版的 UNIX 系統中，如果沒有特別買其專屬的編譯 器的話 (其價格往往不便宜)，人們通常就選擇安裝 GCC 來使用。而且，儘管 GCC 是自由軟體計畫開發出來的，但其所編譯出來的程式品質並不輸給商業版的編譯器， 甚至在某些平台上所編譯出來程式有更好的執行效能。

除了各種編譯器以外，GCC 還內含了其他的工具程式以及各程式語言所需的函式庫，像是 C++ (g++) 所需的 libstdc++, 以及 Java (gcj) 所需的 class 函式庫 libgcj 等。而其工具程式中，最重要的就是 cpp。它是程式編譯時期的前置處理器。通常一個程式在編譯時的步驟如下：

巨集前置處理 → 程式碼編譯與最佳化 → 組譯 → 連結函式庫 → 可執行檔

而 cpp 的工作就是負責整個流程的第一個步驟。通常我們在寫程式時會定義許多巨集 程式碼，同時也會函入若干「標頭檔」(header files, 附檔名爲 .h)，這些用以方便 程式撰寫的手法在編譯時期都會交由 cpp 處理，在原始碼中全部展開後，才交由編譯 器進行實際的編譯工作。當然，這些在呼叫 GCC 進行編譯時都會自動進行，故我們不 需要親自呼叫 cpp。

目前一般的 GNU/Linux 與 FreeBSD 所通行的 GCC 版本爲 2.95.X，而最新版 GCC-3.0 已在不久前正式發表。比起目前通行的版本，新版多了許多新的特色，包括程式最佳化 的改進，新程式語言的正式支援 (如 Java)，新硬體平台的支援與最佳化 (如 IA-64)， 以及新演算法的實作以改進編譯的品質 .... 等等。目前各 GNU/Linux distribution 與 FreeBSD 都在積極整合 GCC-3.0, 在不久的將來我們將在這些新版的套件中使用到 它。

6.3 各種程式工具 binutils

在程式的編譯過程中，經過編譯器 (如 GCC) 的編譯與最佳化之後，其輸出往往就是程 式的組合語言碼。這時後還需要經由組譯器 (assembler) 的組譯動作，將組合語言碼 翻譯成該機器所能接受的機器語言，其輸出的檔案通常稱之外目的檔 (object file， 副檔名爲 .o)。而到最後一步，再

經由連結器 (linker) 將此程式所有的目的檔與其所 需的系統函式庫與啓動模組都連結進來 (可以是靜態或動態連結) 之後，才能產生一個 完整的可執行程式。

因此，如同 cpp 一樣，組譯器與連結器在整個程式編譯過程中扮演著關鍵的角色，但 因爲它們大部分時候都是在幕後工作，因此我們顯少會留意到它們的存在。而 GNU 計 畫旗下的 binutils 套件中，就提供了這兩支重要的程式，分別爲組譯器 as 與連結器 ld。除此之外，此套件還包含了其他工具程式，可以幫助我們管理編譯過後的程式與目 的檔，這些工具包括 :

1. addr2line: 可將內含除錯資訊的程式中某個特定位址轉換爲其原始程式碼的檔名以 及行號，其功能有點像除錯器 (見下一節)。

2. ar: 可將許多檔案打包成一個單一包裹檔案，同時也可以進行包裹檔的修改、資料 節取、刪除 .... 等等的管理工作。它最常被拿來製作靜態函式庫 (static library)，也就是將原始程式碼經由編譯器編譯出一個個目的檔後，就可以 使用 ar 程式將這些目的檔全部打包起來，而這個打包出來的檔案就是一個 靜態函式庫，通常其副檔案即爲 .a。

3. c++filt: 由於 C++ 語言可以藉由函式 overload 的方式，讓許多不同的函式使用 相同的函式名，而這些不同的函式在編譯過後就以不同的組合標記 (assembly label) 來表示，此對應過程稱之爲 mangling。而工具程式 c++filt 所做的 卻是相反的動作 : 它可以由組合標記反向對應出其原來的函式名稱，以便連結 器參考使用，而此反向過程即稱之爲 demangling。

4. gprof: 用來顯示程式的效率測試資訊 (profiling)。通常在我們要試圖提升程式的 執行效能時，我們必須要先知道程式的各部分到底花了

多少時間來執行，如此 我們心裡才能有個底，知道應該由那些部分先下手改進。然而，光從程式碼本 身有時候很難評估到底那個部分最花時間，這時就可以靠這個工具程式幫忙了。 在測示前，首先必須在程式編譯時加入效率測試資訊 (以 GCC 爲例，編譯時必 須加入 -pg 的參數)，之後當我們執行程式之後，就會跑出一個 gmon.out 的 檔案，該檔案即內含了效率測試的結果，但它的格式是不可讀的，這時就必須 用 gprof 將它讀出來。

5. nlmconv: 此工具程式可以將目的檔轉換成 NLM (NetWare Loadable Module) 的格式。

6. nm: 此工具程式可以列出目的檔或函式庫中內含的物件名稱，包括所有的函式名稱與 全域變數的名稱 .... 等。

7. objcopy: 此工具程式可以拷備、轉換不同格式的目的檔。

8. objdump: 此工具程式可以列出目的檔內含的資訊。

9. ranlib: 此工具程式可以產生包裹檔的內容索引，並加入該包裹檔中。此包裹檔即爲 ar 程式所產生的包裹檔。通常在產生靜態函式庫時需要用到它。

10. readelf: 此工具程式可以顯示 ELF 格式的目的檔內含的資訊。

11. size: 此工具程式可以列出目的檔或包裹檔中各節區的大小。

12. strings: 此工具程式可以列出檔案中任何可印出的字串。

13. strip: 此工具程式可以將目的檔或可執行檔中不必要的標記與資訊抽離，讓該目的 檔或可執行檔的大小縮小。這些可抽離的資訊通常是除錯資訊以及其他的符號 標記。

6.4 除錯工具 gdb

在程式的發展過程中，我們往往需要進行大量的除錯 (debug) 工作，以驗証程式的可 靠性與正確性，並盡可能修正所有已知的錯誤。而這類的除錯工作往往是程式開發過 程中最爲繁重的部分，尤其當程式越寫越複雜時。因此，一個好的除錯器 (debugger) 是不可或缺的工具，特別是對於經過編譯之後才可執行的程式而言。經由它，我們可 以直接透視程式的內部，監看程式的運作過程，藉此找出可能的毛病所在。

一般而言，一個除錯器可以做到以下四件事 :

1. 啓動您的程式，並設定任何可以影響您程式運作的條件，用以協助測試程式。
2. 讓您的程式在某個特定條件點下暫停執行。
3. 讓您可以在您的程式暫停執行時，檢視它內部發生了什麼事，像是各變數的值、 其狀態有沒有如同我們的預期 .... 等。
4. 讓您可以改變程式目前的狀態，讓您實驗看看在此改變之下繼續運作會發生什 麼事情。

在自由軟體世界中，標準的除錯器就是 GNU 計畫所發展的 gdb。它支援了多種程式 語言的除錯功能，諸如 C, C++, Fortran, Java, Chill, assembly, 與 Modula-2 等都在支援之列，其中最完整的就是 C 與 C++, 其次是 Chill 與 Modula-2，至於 其他語言的支援則持續在改進中。可以這麼說，凡是 GCC 編譯器家族所支援的程式 語言，其除錯的部分或多或少都以使用 gdb 來完成。

與其他常見的編譯器 /除錯器的運作方式一樣，當我們要使用 gdb 來進行程式除錯 之前，我們必須下特別的參數給編譯器 (對於 GCC 家族

而言，此參數就是 -g)， 讓編譯器在編譯程式時加入一些除錯的資訊，如此編譯完成的程式在 gdb 下除錯 時， gdb 才能從中解析出每個程式位址所代表的變數名稱，以及它們在程式的原始 碼中的位置，以方便我們的除錯工作。

由此可見，除錯的工作並不是單方面的，事實上必須編譯器與除錯器互相配合才行。 如果編譯器的編譯結果內含了不適當的除錯資訊，則除錯器仍然無法對該程式進行 除錯的工作。這也是爲什麼 gdb 對於其他語言如 Fortran 等的支援仍不足的原因， 除了 gdb 本身要改進，編譯器那邊也需要配合改進才行。除此之外，編譯器在編 譯程式時，其產生目的碼的方式也直接影響了程式能否在除錯器中除錯，例如對於 Chill 與 Modula-2 而言， gdb 目前只能接受來自 GCC 家族編譯器所編出來的程式 碼，至於其他的編譯器則還沒有辦法。

gdb 是完全在文字模式下運作的，它擁有許多命令可以方便我們的除錯工作。然而 對於已用慣了圖形介面的使用者而言還是不太方便，因此有許多圖形介面的除錯器 便接二連三地開發出來，但它們絕大部分都還是以 gdb 爲核心來進行除錯工作， 它們不過是在 gdb 之上加一層方便操作的外衣，當我們操作這層外衣時，它將我 們希望做的動作告訴 gdb, 而 gdb 工作完後再將結果送出來，顯示在這層外衣上。

這類的圖形介面除錯器中，最俱代表性的就是 ddd, 它的外衣是以 Motif/Lesstif 函式庫開發出來的，整個操作介面相當漂亮而且視覺化。此外，我們稍後將談到的 整體開發環境 (IDE) 也是其中之一。

6.5 編譯自動化 make

當我們的程式寫得很龐大時 (可能包含許多原始程式檔案)，往往需要相當複雜的編譯 動作才能將程式整個編譯完成，並且安裝到正確的位置上

使用。這時候，光靠手動執 行每一個編譯指令顯然是很沒有效率的。同時，由於一個大型程式的編譯往往要花很 久的時間，故如果每次程式一有修改 (不論修改的幅度有多少) 就要整個從頭編譯， 也是相當沒有效率的。

解決問題之道，就是使用 make 程式，將整個程式編譯的工作自動化。當此程式啓動 時，它會讀入目前工作目錄下的指令檔 Makefile，並根據該檔案的指令一步步地執行。 因此，我們就可以將整個程式編譯的細節寫在 Makefile 中，當程式需要編譯時就執 行 make 將它編譯完成。

不儘如此，我們還可以在 Makefile 中指明各編譯產物與其來源檔案之間的相依關係， 舉個例子 : 例如我們要發展一個靜態的函式庫，其最終產物是一個 .a 的函式庫檔， 而該函式庫檔是由各別的目的檔所組成的，而各別的目的檔又是由各自的原始程式碼 編譯而來。如此，我們就有了一連串的相依關係了。當我們在 Makefile 中指明了這 些相依關係後，如果有任何一個原始程式修改了，則當我們執行 make 時，它只會去 編譯與該原始程式檔有相依關係的目的檔，如此一來該目的檔就等於修改過了，由於 最後的函式庫檔是相依於所有的目的檔，故 make 最後會再重新產生一個函式庫檔案， 如此即編譯完成。

換句話說， make 依據從 Makefile 所指定的相依關係，來判斷編譯過程中各中間產物 是否需要重新編譯，而判斷的標準就是該中間產物的來源檔案是否有被修改過，也就是 來源檔案的最後修改日期是否較該中間產物來得晚？是的話就表示該來源檔案已修改過 了，故該中間產物需要重新編譯。因此，每當原始程式有小幅的修改時，它就只會去重 新編譯眞正修改過的部分，而不會每次都從頭開始編譯，如此效率就可以大符提升了。

在 GNU 計畫中，同樣也提供了一個強大的 make 程式，它除了具備上述的基本特性之 外，同時還包含了許多 GNU 的延伸功能，例如各種

巨集的定義，條件判斷式 .... 等 等，而有些延伸功能是其他版本的 make 所沒有的。因此，有許多大型的自由軟體， 都會特別指明要 GNU make (或者還包括 GCC 及其他的 GNU 編譯工具) 才能進行編譯， 因爲它們的 Makefile 中已隱含了 GNU make 許多特有的功能。

6.6 跨平台輔助工具

在一個系統上開發一套軟體是一回事，而要將此軟體整個移到另外一個不同的平台上 編譯、執行又是另一回事。因爲不同的平台有不同的特性、不同的開發環境、可能連 相同名稱的工具程式的用法也不相同。因此，要將程式跨平台移植不是一件容易的事， 甚至連相似程度最高的各 UNIX 系統間也是如此。

然而，人們對於自由軟體總是情有獨鍾，因爲它不但可以免費取得，而且還可以自由 使用，甚至連原始碼都有。因此，每當人們看到一套自由軟體在某個系統平台上出現 時，就自然而然會去想 : 那它能不能移植到我現在用的系統平台上？故「跨平台移植」 這個議題在自由軟體世界就顯得格外迫切需要。

但「跨平台移植」不是件簡單的事，因爲每個系統都有差異，而這些差異有些是可預 期的 (或固定的)，有些則不能預期。是否有方法可以達到完全的自動化呢？因此， 這邊就有兩種不同的解決方案出現 :

1. 各平台提供其平台的資訊讓編譯系統參考，如些編譯系統在進行程式編譯時， 可以依程式原始碼中內含的規則做適度的調整，以適應不同的平台。

2. 程式原始碼主動出擊，在編譯之前自行偵測此系統平台的組態爲何，然後動 態調整自己以適應此系統平台，讓編譯工作得以順利

進。 其中 1. 方案在許多大型的開發環境中較常見，例如 X Window 系統或 gtk+ 函式庫 等。而 2. 方案則是 GNU 的漂亮傑作。很多時候，我們也可以同時採行這兩個方案， 相輔相承。以下，我們就舉幾個重要的例子，來看看這些跨平台的解決方案。

- gtk-config

  這是 gtk+ 函式庫的跨平台輔助工具，它是一個標準的 gtk+ 工具，意味著不論任 何系統平台，只要有 gtk+ 函式庫與程式開發環境就會有它，而它就是用來提供 gtk+ 在此系統平台的組態資訊，也就是前述的方案 a., 包括程式編譯時要下什麼 參數給編譯器、一些檔案路徑名稱、所需連結的函式庫名稱等。

  不只是 gtk+, 很多新近設計的大型函式庫也採行這種解決方案。

- xmkmf/imake

  xmkmf 是 X Window 系統下的標準工具之一，它是用來將 Makefile 的樣本檔 : Imakefile 轉換成此 X Window 的開發環境下可用的 Makefile。由於 Imakefile 只是一個「樣本」檔，不是眞正的 Makefile, 故不能拿來直接讓 make 使用。它 只包含了如何編譯原始程式的簡單資訊，以及個原始碼與中間產物的相依關係， 但於系統平台相關的細節則一概不提。而這些相關細節就交由 xmkmf 來補足，故 它的解決方式也屬於方案 a.。當 xmkmf 讀入 Imakefile 後，它會將此平台上的 X Window 系統所有細節都列出來，一起寫入 Makefile 中，如此編譯系統與 make 就會根據此 Makefile 來正確工作。

  事實上， xmkmf 只是一個簡單的 shell script (詳見後述) 而已，它眞正運作是 靠 imake 程式來完成。 imake 可以讀入 Imakefile, 以及其他的組態檔、設定檔、 樣本檔 .... 等等，合併之後產

生 Makefile 輸出。這裡所讀入的檔案中，除了 Imakefile 是原始程式提供的以外，其他的都是屬於 X Window 的檔案。由於 X Window 系統已在許多硬體平台上有移植，故它內部包含了各系統平台所需的組 態檔與設定檔，至於到底那一個才會生效，則視當初建立此 X Window 系統時的 系統組態而定。 而以下三個工具 : autoconf, automake, 以及 libtool 則全部都是 GNU 計畫的傑作， 它們都是採用方案 2.: 主動偵測系統平台的組態然後調整自己，以適應該平台的開發 環境。這三個工具互相搭配，相輔相承，以下我們分別作簡介。

- autoconf

  autoconf 是最基本的軟體偵測系統平台的工具。此套件收集了大部分的 UNIX 系統 (可能還包括了其他作業系統) 平台的基本特性，經過分析歸納成幾個基本的原則， 然後再根據這些原則來測試系統，並依據測試結果來調整原始碼本身，最後才開始 編譯工作。

  通常當我們將程式寫好後，我們可以依據程式的需求來加入 autoconf 的支援。例 如要編譯我們的程式可能需要某些函式庫、可能需要額外的工具程式、可能對編譯 器本身的特性有特別的需求 .... 等等，這些需求都是我們事先可以知道的，但我 們不能確定想拿我們的程式來編譯的系統平台是否能符合這些需求，故我們就將這 些需求寫好，經由 autoconf 程式將它轉換成一個名爲 configure 的 shell script, 如此一旦有人要編譯我們的程式時，我們會請他 /她先跑一次這個 configure 的 shell script, 它就會根據我們程式的需求去一項項地測試系統，並將測試報告以 各種方式輸出。輸出方式可能包括產生 config.h 的標頭檔，讓我們的原始碼可以 利用，如此我們的原始碼就會知

道這個系統有什麼？沒有什麼？而在編譯開始時告 訴編譯器要編譯那幾段的程式碼。或者 configure 也可以根據我們事先準備好的 樣本檔 Makefile.in 來產生可以適應該系統平台的 Makefile, 告訴 make 在編譯 時應該下什麼指令給編譯器？如何安裝編譯好的程式 .... 等等。

也許有人會覺得奇怪， make 程式不都有支援 Makefile 中使用條件判斷式嗎？我們 爲什麼不直接將適應各種系統平台的資訊寫入 Makefile 的條件判斷式中，讓它在 編譯時可以直接適應系統平台，而非得用 autoconf 與 configure 不可呢？原因在 於，各家系統平台的 make 程式所能接受的條件判斷式的語法都不同，因此我們很 難寫出一份所有平台都能接受的 Makefile。然而， shell script 的基本語法卻是 可以在各家系統平台 (至少各家 UNIX 平台) 中通行的，而且就算只用到基本語法， 也足以完成所有的系統測試工作，這就是爲什麼拌隨程式原始碼而來 configure 程 式是採用 shell script 寫成的原因了。

- automake

  automake 可以用來產生符合 GNU 編程標準 (GNU coding standard) 的 Makefile 的工具程式。此標準制定的目的是希望所有的 GNU 程式與系統都可以達到乾淨實作、 具有一致性、以及容易安裝等特性，同時也能做爲程式的跨平台移植方面的指引。

  對於大型的程式而言，若要達到上述的要求往往需要耗費不少心力在程式設計上以及 Makefile 的撰寫上，尤其是牽涉到跨平台的問題時。因此，我們就可利用這個工具 程式來簡化 Makefile 設計上的麻煩。它必須與 autoconf 搭配使用，因爲最後當程 式要在不同平台下編譯時，仍然需要 autoconf 的 configure 程式來偵測系統，以 做調整適應。當我們使用

automake 時，我們並不是用它直接產生萬用的 Makefile, 而是將我們程式的編譯需求編寫成一連串的巨集指令，由 automake 將它轉換成 Makefile.in 檔案，在這轉換過程中 automake 就會在其中加入可以與 autoconf 互 相搭配的指令。而當程式要進行編譯時， configure 就會以此 Makefile.in 爲樣本 產生眞正的 Makefile, 供編譯時使用。

可以這麼說，使用 automake 可以幫助我們簡化撰寫 Makefile 的樣本檔的麻煩，當 然我們也可以不使用它，只是這時候就必須自己設計此樣本檔了。

- libtool

  在跨平台移植的問題上，函式庫的製作也是一個麻煩，特別是在製作可分享函式庫 (shared library) 方面，因爲在各家的 UNIX 平台、各家的編譯器與連結器下，所 需給的參數都不一樣。就算採用 autoconf 也難以通盤解決這些問題。因此， GNU 計畫就特別爲此發展出一套 libtool 的工具程式。

  如果 autoconf 一樣， libtool 內函了各系統平台上製作函式庫的資訊，並將這些 資訊整合到一個統一的使用介面下，使我們在撰寫時不必去擔心各平台間的差異問 題。不只如此，它同時還考慮到了程式使用動態模組的移植問題，因爲有些平台支 援標準的動態模組呼叫介面 (內建於系統的 libc 或其他函式庫中)，有些則採用其 內定的標準，有些則完全不支援。很顯然如果我們的程需要使用動態模組時，就不 具可移植性了。然而使用 libtool 則不會有這些問題，因爲 libtool 還提供了一 個支援動態模組的函式庫原始碼，我們可以將它整合到我們的程式中，如遇到了不 支援動態模組的系統平台，就可以編譯這個函式庫原始碼來使用。

libtool 可以整合到 autoconf 與 automake 中，一起搭配使用。當程式碼要進行 編譯時， configure 程式在進行系統偵測時也會呼叫 libtool 來進行系統的函式庫 編譯支援的偵測，並將偵測結果一併回報給 configure, 以調整原始碼對系統的適應。

6.7 程式維護 cvs

在程式開發的過程中，我們便要面臨一連串的維護問題，例如我們可能會爲了修正程式 的錯誤、加入新功能而不斷修改程式碼，我們可能需要做程式版本控制，或爲了追蹤、 記錄整個開發過程而要保留許多不同版本的程式碼 .... 等等。尤其當我們的程式越寫 越大時，這些問題將越形嚴重。爲了解決這些問題，常久以來就開始有各種程式碼版本 控制與維護工具的發展，例如早期在 UNIX 系統上常見的 SCCS 與 RCS 等。

cvs 則是繼這些傳統的維護工具之後，新一代的程式開發與維護系統，目前已廣泛受到 各軟體開發計劃與專案的採用，它擁有以下的特色 :

1. 程式碼各版本的儲存與維護 : 一個開發中的程式每一次的修改就會有一個版本號碼， 但其程式碼只會在 cvs 的內部資料庫 (Repository) 中保存一份，而程式碼後續的 修改只會在資料庫中儲存修改的部而已，並不會爲每個不同的版本都存一份程式碼， 如此可以有效率地利用儲存空間。

2. 程式碼的版本追蹤回溯 : 我們隨時可藉由 cvs 回溯程式碼各版的開發過程與改變狀 況，同時每一次的修改在進入 cvs 的資料庫時，都會留有完整的記錄以供查詢。如 此當我們程式在發展一段時間後，若突然冒出了問題，可以很容易地回溯問題的根源 並解決它。

3. 程式碼版本的分合控制：程式的發展有時會因爲需要而分成數個支派分頭開發，或 者將數個支派的程式碼融合後繼續往下發展，這些在 cvs 系統中都提供了完整的支 援，以配合我們調整各種開發方向。

4. 支援多人合作開發專案：當同時有多人合作撰寫同一個程式時，管理的問題可能就 會浮上台面。例如某甲已寫好了某一段程式，並將程式碼送回公共的資料庫中，但 某乙可能同時間做了程式的修改，也將修改的部分送回公共的資料庫中，很顯然的 二者所做的修改沒有同步，更嚴重的情況可能是如果某甲與某乙之間沒協調好的話，則某甲所寫的那一段因某乙將他的修改送回來後，被覆蓋掉而不見了。

   這時，使用 cvs 就能幫助我們解決這些麻煩的協調與管理問題，當某甲與某乙同時 修改了同一份程式碼，並送回給 cvs 時， cvs 會記錄他們所修改的部分，同時試圖 將他們的修改合併起來，如果無法合併，則它會在程式碼中著明可能有問題的部分， 靜待他人將程式碼取回後解決。而當有任何人將修改後的程式碼送回給 cvs 時，它也會以適當的方式 (如寄發電子郵件，或公告在網頁上) 通知所有的參與開發者，請 他們更新他們的程式碼，與 cvs 保持同步。

5. 程式碼遠端管理維護：cvs 也可以透過網路進行程式碼的截取與維護工作，而不需要 將所有的參與開發者都鎖在同一部機器裡。如此可以方便世界各地的人們透過網路參 與程式開發專案，這也是促進自由軟體開發模式的因素之一。

6. 程式碼匿名截取：在一般情況下，只有參與開發者可以將他們修改的程式送回給 cvs 保存，但對於其他人而言，他們可能急於測試目前開發中版本的功能，或者只是好奇 目前這個程式到底新加了那些

特色，因此他們可以等不及這些程式開發者釋出新版程 式出來，而直接經由 cvs 的匿名程式碼截取的功能挖出最新的程式碼來玩。

由於整個程式的開發與維護都是在 cvs 中進行的，故理所當然的 cvs 中的程式碼 就是最新的程式碼。 而站在自由軟體開發者的角度，開放 cvs 的程式碼匿名截取也有許多好處，其中最 大的好處就是讓更多的人參與測試。必竟參與開發的人是有限的，他們所能進行的程 式測試也只能到某個限度，故如果能在程式開發過程中讓更多有興趣的人們幫忙測試， 對於程式碼的除錯與品質提升有相當大的幫助。

6.8 各式各樣的直譯式語言

之前我們在談程式開發時，我們都把焦點擺在需要編譯的程式上面，例如 C, C++, Fortran, Java .... 等等。因爲程式需要編譯後才能用，故就會牽涉到編譯器、連結器、 函式庫 .... 等等各式各樣的跨平台問題，等著我們去傷腦筋。

然而，另外有一類的語言所寫出的程式是不需要「編譯」的，稱之爲「直譯式語言」， 這類語言只需要一個直譯器 (interpreter) 來做程式碼的翻譯，就可以完成我們想做的 工作。故使用直譯式語言顯然就單純許多，比較沒有跨平台方面的問題。

然而就這個編譯式語言與直譯式語言的差別，就明顯指出了二者所適用的場合。編譯式 語言因爲是經由編譯器將程式碼轉換成機器直接可接受的機械碼，因此它的速度可以很 快，同時我們也可以很容易地調整程式碼來實作我們需要的功能。然而它的缺點是程式 寫起來往往很麻煩，以 C 語言爲例，它算是一個中階程式語言，因爲它每一段程式碼都只是命令電腦做一件簡單的動作，如果沒有方便的函式庫幫忙的話，如果我們想要實作 一個稍微複雜一點的功能，程式寫起來往往就會很長。

但直譯式語言則不同，它是靠該語言的直譯器來做程式碼的翻譯，並由直譯器代替程式 碼本身來執行要做的所有的動作。可想而知，這樣子的執行效率往往是不及編譯式語言 的。然而直譯式語言通常有一個很大的特點，就是它們會將一些經常需要的複雜動作用 一兩個簡單指令包裝起來，如此我們就可以用很簡單的程式碼來做到一些複雜的功能。 故使用直譯式語言可以大符簡化程式的複雜度，縮短開發的時間。然而，萬一直譯式語 言不支援某項複雜功能時，我們還是得回到編譯式語言來開發。

因此，當我們需要講求彈性與執行效率時，我們會選擇用編譯式語言來開發。若我們講 求方便性，並希望避開麻煩的跨平台移植問題時，我們會選擇用直譯式語言。事實上， 在很多程式開發專案中，通常是這兩類語言並用的，最明顯的例子就是我們在前面提過 的跨平台工具。我們用編譯式語言寫成的程式本身，要移到其他平台上編譯時，我們需 要用 autoconf 所做出的 configure 程式進行系統測試，這個 configure 就是用 shell script 這個直譯式語言寫出來，正因爲它沒有跨平台的問題，所以可用來做編譯初期的 系統測試，而讓更具執行效率、更有威力的編譯式語言程式可以順利安裝成功。

以下，我們就介紹幾個自由軟體世界中常見的直譯式語言，這些語言各自有其特性，我 們可以依我們的需求來選用。

1. shell

   shell 原來的設計目的並不是用做一個程式語言。它其實是「命令解釋器」，它等待 使用者輸入指令，解釋指令的內容，並呼叫指令所指定的應用程式來完成工作，其地 位就好像 Microsoft MS-DOS 系統中的 COMMAND.COM 一樣。這種將命令解釋器與系 統底 層核心分開的實作方式，是 UNIX 系統的一項創舉，如此使用者就

可以依其使 用習慣而選用不同的 shell 來做爲其人機操作介面，而 Microsoft MS-DOS 的 COMMAND.COM 也是仿效這樣的設計。

爲了方便指令的輸入與進階的操控功能， shell 進一步地發展出了可程式化的特色， 我們可以將一連串的指令依 shell 的語法寫在一份文稿 (shell script) 中，而直 接執行這份文稿就可以一口氣處理掉所有的工作。因此，這個 shell script 文稿就 成了直譯式語言，而此語言的直譯器就是 shell 本身。剛開始此語言只包含很簡單 的功能，但發展到後來漸漸成爲一項功能齊備的語言，包括條件判斷式、迴圈、檔案 輸出入重導向與檔案狀態類形的判斷 .... 等等。

shell script 語言的語法依 shell 的不同而有不同，但在這許多種類的 shell 中 大至上可歸類成兩類：一是 Bourne Shell (sh) 類，包括 sh, ksh, 與 GNU 計劃所 發展的 Bourne Again Shell (bash)。另一類是柏克萊 UNIX 所發展的 C Shell 類 (csh)，包括 csh 與 tcsh，其最大的特色是它的語法很接近 C 語言，這讓已熟悉 C 語言的使用者可以立即上手。除此之外，還有同時擁有 Bourne Shell 與 C Shell 的特色的 shell, 其中以 zsh 爲代表。

這許多 shell 中，以 Bourne Shell (sh) 最爲通行，幾乎各家各派的 UNIX 系統都 有這個 shell，可以說是 UNIX 世界中的標準。它所支援的語法相當簡單，但足以應 付許多簡單的程式需求。幾乎所有的 UNIX 系統管理工作都是由 sh 的 shell script 達成的，連 autoconf 的 configure 程式都只用到 sh 的語法而已。而之後發展的 ksh 與 bash 則是以 sh 爲基礎再做擴充，可以支援更多的程式語法與較複雜的資料 結構，以及許多功能，其中尤以 bash 的實作最爲完備。在另一方面， tcsh 也是以 原始的 csh 爲基礎重新實作出來的，它也擁有與 bash 相匹敵的程式語法與功能。

前面我們已提到，直譯式語言可以經由幾個簡單的指令完成很複雜

的工作，而 shell script 達成此功能的方式就是直接呼叫並執行應用程式。因此，在標準的 UNIX 系 統中，都會有一組標準的 shell 工具程式，供我們撰寫 shell script 或其他方面 使用。這些 shell 的工具程式每一個都有特定的功能，而且我們還可以使用管道 (pipe) 的方式將許多工具程式串接起來，將一個的輸出導入另一個的輸入來處理， 如此便能組合出許多原本沒有的複雜功能。這些 shell 工具程大都以檔案與字串的 處理爲主，例如字串搜尋的 grep, 字串代換的 sed, 字串編輯的 ed, 字串解析的 awk .... 等等，其他還有許多。

而在字串處理方面，還有一種特別的語法可以做字串的搜尋、解析、與替換等的工 作，此語法稱之爲「正規化表示式 (regular expression)」，這在許多 UNIX 的工 具程式都有支援，包括上述的 shell 工具程式、許多文字編輯器、以即將要談到的 其他直譯式程式語言，甚至在系統底層函式庫 libc 中也有它的呼叫介面，其使用 相當廣泛。

然而，光使用 shell script 與其附帶的 shell 工具程式，所能處理的事情仍然很有 限，例如它就缺乏複雜的資料結構，缺乏複雜的數學運算功能，也難以做細部的檔案內 容解析 (如眞要這麼做， shell script 寫起來可能會很長)。因此，就有許許多多的直 譯式語言開發出來，以下就幾個較常見的稍作簡介。

2. perl

   perl 是一個非常強大的字串與文字檔案處理語言，它結合了 C 程式語言與許多 shell 工具程式 sed, awk, sh .... 等的特性，以方便程式開發爲第一考量。因此 它不像其他的程式語言那樣，只包含了最基本的語法與元件，而其他複雜的功能都 必須由這些基本的元件堆積出來。相反的， perl 的說明文件常常會見到一句口號 : 「您常常可以想到新的語法來完成您想做的事」。

perl 的應用相當廣範，舉凡網頁 cgi 程式的撰寫，複雜的系統管理與程式套件的 管理，甚至許多應用程式本身的原始碼 (如 automake 就是以 perl 寫成的)，在 在都見得到它的蹤跡。

3. python:

   python 是一個物件導向式的直譯式語言，與 perl 相較，會發現它的程式碼寫起 來乾淨許多。它擁有明確的語法，有許多外掛模組、類別 (class)、以及高階抽像 的動態資料結構，我們還可以很容易地用 C 或 C++ 語言來寫新的外掛模組，以增 強 python 的功能。

4. tcl/tk:

   tcl 語言是 John Ousterhout 教授當年在加州柏克萊大學任教時開發出來的語言， 而 tk 是 tcl 的延伸，它可以與 X Window 圖形介面函式庫結合，以簡單的直譯式 語言來寫圖形介面的應用程式。有許多 X Window 的小程式就是以這個語言完成的， 最常見的一個例子就是 Linux kernel 在編譯時的圖形化設定介面：make xconfig。

5. php:

   php 是專爲網頁設計的直譯式語言。傳統的網頁是以 html 語言寫成的，它本身也是 一個直譯式語言，而它的直譯器就是我們的一般使用的網頁流覽器。然而當我們要設 計複雜的網頁或要做到其他特殊功能或特殊效果時，光使用 html 就很不方便了。這 時候， php 語言就可以派上用場了。它是一個網頁伺服端、跨平台、且內嵌 html 的 語言，同時它也有許多外掛模組，可以與許多資料庫 (如 mysql) 連在一起，以方便 我們建置全功能的網頁服務。

6. javascript:

   javascript 是由 Netscape 所開發的直譯式語言，可以讓我們設計互動式的網頁。 儘管它具備許多與 Java 語言相同 (或相類似) 的特

色與結構，但它卻是完全獨立 設計出來的。 javascript 可以與 html 語言交互作用，讓我們在網頁上加料，以 達到動態網頁資料的效果。 javascript 目前已廣泛地得到許多軟體廠商支援，同 時它是一個開放的語言，任何人都可以使用它而無須購買它的版權。目前在許多網 頁流覽器上也到處都有它的支援，例如 Netscape 與 Microsft Internet Explorer 等。然而， Microsft Internet Explorer 只支援了一部分，而該部分被他們稱之爲 Jscript。

6.9 程式文件維護

好的程式，需要有好的文件說明，使用者才曉得如何去使用它，發揮它的功能，以完成 所需的工作。在傳統的 UNIX 系統中，其線上文件系統就是 man page, 當我們執行 “man ¡程式名、設定檔名、或函式名 ¿” 就會列出它們的說明文件供我們流覽。

隨著時代的變遷，有越來越多的文件格式出現，包括高品質的排版系統 TeX/LaTeX 與 文件輸出 Postscript/PDF、多媒體超連結網路文件 html、以及 GNU 所開發的文件閱 覽系統 info page .... 等等，使得 man page 在某些方面已有些不方便，或不合時宜 了。然而，在不同場合我們會希望使用不同格式的文件，例如在一般的文字終端機下還 是 man page 或 info page 比較方便，但如果要列印時我們希望用 Postscript/PDF，而放在網路上供人流覽當然是用 html。難道我們要分別爲這些不同的格式都準備一份 文件嗎？

爲了解決這些困擾， GNU 計畫就特別開發了一套文件準備系統 texinfo，它可以讓我們 只寫一份原始的文件檔，然後透過 texinfo 以及其他週邊程式 (如 TeX/LaTeX) 將它 轉換成許多其他格式。幾乎所有的 GNU 計畫旗下的軟體的文件都是由 texinfo 寫成 的。

除此之外，還有一套稱爲 cweb 的工具，它主要用於程式源始碼的注解上。我們在寫程 式時經常會在程式碼的字裡行間加入注解，以方便他人了解這段程式是做什麼的，同時 也避免自己日久後忘了程式爲什麼要這樣寫。這些程式碼的注解往往是零零散散的，而 它的詳盡呈度與撰寫程式者的習慣與風格有很大的關係。然而，如果使用 cweb 來幫助 撰寫這些注解，由於 cweb 有一定的格式規定，故當我們依照這樣的格式來填寫注解時， 自然會達到一定的詳實呈度。而且， cweb 還有解析程式語言如 C/C++ .... 等的功能， 它能自動判斷那部分應該要加注解，而提示我們加入。自然而然地，這些詳實的注解就 可以成爲本程式碼的「說明書」，而 cweb 還可以進一步將這些注解自程式碼中抽出來， 經周邊工具程式的排版、轉換，而眞的轉換出 Postscript/PDF 或 HTML 等格式的「說 明書」出來。

還有其他相關的文件準備系統，如 GNU/Linux 的 HOTWO 文件常用的格式 SGML，以及 近年來漸漸流行的 XML 格式 .... 等等。但有關它們的說明已超出本文的範圍，有興 趣的朋友不妨參考文末的參考文獻。

6.10 整合開發環境與其他開發環境

以上我們已介紹了許許多多的程式開發工具，有經驗的程式設計師曉得如何搭配使用它 們，以方便程式開發工作。因此他們在實際撰寫程式時，往往不像在 Microsoft 系統 平台下那樣，要啓動一個大型的整合開發環境 (Integrated Development Environment) 才能開始寫程式 (例如，最著名的就是 Borland 公司所發展的 Borland C/C++, C++ Builder .... 等)。事實上他們在寫程式時，多半就直接開一個簡單的文字編輯器 (如 vi) 就可以了，了不起就在 X Window 環境下，多開出幾個終端機視窗，

這個視窗用 vi 編寫程式，那個式窗參考其他文件或原始碼，下一個視窗就用來執行編譯指令 make 並 測試程式，再下一個視窗則跑一個 gdb 來做程式除錯 .... 等等。因此，對他們而言， 並不特別需要一個整合開發環境，光以上的工具就綽綽有餘了。

但在某些場合下，能有一個整合開發環境，將上述的程式發展工具整合起來，在同一個 使用者介面下使用還是相當方便的。在自由軟體的世界中不乏這類的整合開發環境，特 別是近來有越來越多的軟體廠商的加入，他們將原本只出現在 Microsoft 平台上的開發 環境移植到自由軟體平台上，使得可用的開發環境更加豐富起來。其中最著名的是日前 Borland 公司移植到 GNU/Linux 平台上的 Kylix 套裝軟體，讓程式設計師可以開發跨 GNU/Linux 與 Microsoft 平台的 Delphi 程式。

而在自由軟體這邊，整體開發環境的發展也在起步階段。在早期有 xwpe，它的樣子看起 來就好像 Borland 公司早期在 MS-DOS 平台上的 Borland C++ 的開發環境，它將 GCC、 文字編輯器、與除錯器 gdb 等整合在一起，讓我們可以方便地在此圖形介面上進行開發 與除錯的工作。另外， KDE 計畫也正在發展一套整合開發環境 kdevel，其使用者介面就 是以 KDE 函式庫寫成的，可以方便我們開發 KDE 的應用程式。

然而，還有另一個較顯爲人知的整合開發環境，就是 GNU 計畫的第一個作品：emacs。 emacs 是一個文字編輯器，但它的功能早已遠遠超越一般的文字編輯器，我們可以在 emacs 中讀電子郵件、把它當作檔案管理員來管理檔案、甚至在經過適當的設定與調 整之後，它就可以將 GCC 等各種開發工具整合起來，成爲一個名符其實的整合開發環 境。比起 xwpe 與 kdevel， emacs 顯然成熟很多，而且使用上更有彈性，很值得有興 趣的朋友仔細玩味。

參考資料：

1. GNU project[85]
2. GCC[6], info gcc
3. binutils[86]
4. gdb[87], info gdb.
5. ddd[88]
6. make[89], info make
7. gtk+[34]
8. xmkmf/imake man xmkmf, man imake
9. autoconf[90], info autoconf
10. automake[91], info automake
11. libtool[92], info libtool
12. GNU coding standard[93]
13. cvs[94], info cvs
14. bash[95], info bash
15. csh/tcsh Using csh & tcsh by Paul DuBois, Publish O'Reilly, ISBN: 1-56592-132-1
16. perl[96, 97], man perl
17. python[98]
18. php[99, 100]

19. tcl/tk[101, 102]

20. javascript[103, 104]

21. texinfo[105], info texinfo

22. cweb[106]

23. SGML/XML[107]

24. emacs[108]

25. xwpe[109]

26. kdevel[110], search for `kdevel`.

27. guide[111]

28. kylix[112]

29. codeforge[113]

30. UNIX Power Tools, 2nd ed., by Jerry Peek, Tim O'Reilly & Mike Loukides, ISBN: 1-56592-260-3

31. Learning the GNU development tools[114]

第七章

程式庫 (Library)

作者：謝東翰

7.1 總論

由於我們的程式中經常需要處理許多特定的動作，例如搜尋與排序、行程管理與通訊、檔案格式轉換、甚至繪圖 .... 等等，而這些動作可能很不單純，若每個動作細節都從要從頭開始設計撰寫的話，往往相當耗時耗力，而且很不經濟。因此，程式設計師便著手將程式中常常需要的一些動作分類、整理，將它們寫成一個個「函式庫」，供程式開發者使用，而不必每次都重新發明輪子。因此，「函式庫」可以說是程式開發工作中重要的寶藏，也許一般使用者並不覺得，但它們也是支持系統中所有應用程式正常運作的背後無名英雄。

7.2 系統呼叫

7.3 標準函式庫

7.3.1 簡介

函式庫的種類相當繁多，而且用途各自不同。以 UNIX 系統而言，其最底層，而且也最重要的函式庫，莫過於 C 函式庫了。由於 C 語言可以說是 UNIX 作業系統的母語，早年當 UNIX 還在 AT&T 草創之初，爲了讓整個系統在各硬體平台間的移植工作更爲容易，同時儘可能地保持系統的執行效能，才因應而開發了此程式語言，並將整個作業系統以此語言重新改寫。目前據估計在一般的 UNIX 系統中約 90% 以上是由 C 或 C++ 程式碼寫成的，其中包括大部分的系統核心、系統函式庫、大部分的系統工具、視窗系統等等皆是。

標準的 C 語言中並不包含如資料的輸出入、記憶體管理、及其他進階的系統服務 .... 等其他程式語言中常見的元件。相反的， C 語言將這些元件留給作業系統來實作，而當我們的程式需要使用這些元件時，必須經由作業系統提供的 C 函式庫 (libc) 來取得這些服務。因此， C 函式庫可以說是除了系統核心以外，所有應用程式賴以執行的基底環境，同時它也是應用程式與系統核心溝通的橋樑。

一般而言，一個標準的 C 函式庫會包含兩類的函式群，一爲與核心溝通的系統呼叫窗口，二爲一般常用的函式物件。「系統呼叫」可以說是應用程式可以由作業系統中取得的最低階操作，是由系統核心直接提供服務，它提供一組系統呼叫介面供應用程式使用，而系統呼叫的入口處就在 C 函式庫的某些函式中。因此，應用程式藉由連結 C 函式庫的這些系統呼叫窗口，進而與系統核心連繫。而一般常用的函式物件則包括字元、字串處理、數學函式庫、搜尋排序演算法、標準輸出入 ....

等等。不論是系統呼叫窗口或是常用的函式物件，爲了達到程式在各 UNIX 平台的可移植性，這中間必須要有一定的規格存在，供系統開發者依循。這些規格日後演生出許多不同的標準，詳見後述。

除此之外，函式庫與應用程式間的連結形態可分爲「靜態函式庫 (static library)」與「可分享函式庫 (shared library)」兩種。前者只是單純將所有的函式與物件打包成一個函式庫檔，而使用此函式庫的程式在編譯時，必須將其用到的所有函式與物件的程式碼自函式庫檔抽出來，然後與程式本身的程式碼連結在一起，如此才能形成一個完整的可執行檔。然而，由於函式庫中內含的函式與物件在許多程式中都會用到，如果每個程式都必須連結一份所需的函式與物件時，顯然相當佔空間。因此，才有後者的「可分享函式庫」的出現。

在支援分享式函式庫的系統環境下，程式編譯時不需要將所需的函式與物件靜態連結進來，而是等到程式要執行時，才由作業系統自動做連結的動作。由於很多程式都會用到相同的函式物件，故事實上作業系統在執行時只會在記憶體中保存一份函式物件而已，當有程式在執行中要用到它時，就自動跳過來執行，執行完畢後再跳回原程式中。因此，同一份函式物件可以讓所有的程式分享，這不僅可節省硬碟的儲存空間，同時也節省執行時整體的記憶體使用量。

所有的 UNIX-like 系統都提供了它們的 C 函式庫，而在新近設計的 UNIX-like 系統中，多半會有分享式函式庫的支援，但其形態可能各異，支援承度也有所不同。一般而言，可能以 UNIX 系統實驗室 (UNIX System Laboratories, USL) 所開發的 ELF (Executable and Linking Format) 格式最爲常見，而該格式也已被認可爲「可移植系統」的標準之一，就連使用其他形態的分享式函式庫的 UNIX 系統中，也開始提供與 ELF 格式相容的使用及呼叫介面。

以下，我們就以 GNU/Linux 系統爲主，介紹其 C 函式庫的規格與

特色。

7.3.2 GLIBC 的規格

在 GNU/Linux 系統中，其 C 函式庫的發展史點出了 GNU/Linux 演進的幾個重要里程碑，由此可以突顯出 C 函式庫在系統中的地位與重要性。早期的 GNU/Linux 系統並不支援可分享函式庫，因此所有的應用程式都是以靜態連結的方式存於系統中。直到 1995-1996 年 libc5 問世以後，系統才開始支援 ELF 可分享函式庫，同時該版的 C 函式庫也實作了其他 UNIX 上大部分的功能與函式群，可謂 GNU/Linux 發展上的一大進步。

然而 libc5 在程式國際化 (I18N) 與本土化 (L10N) 方面的支援很差，故那個時候若要開發所謂中文化的程式，就非得自行實作一套標準不可。直到一兩年後， GNU/Linux 換用了 GNU 所開發的 glibc-2.0 做爲其 C 函式庫後，其國際化與本土化的支援才開始起步，後來歷經 glibc-2.1，到了現在的 2.2 版，整個多國語文的開發環境才大至成熟。

採用 glibc 做爲系統的 C 函式庫，是 GNU/Linux 演進的一個重要里程碑。在 GNU 的計畫中，開發 glibc 原本是要給他們尚未問世的核心 GNU/Hurd 用的，由於當時幾乎 99% 的 GNU 系統工具已開發完成，就獨缺核心 Hurd，而恰巧 Linux 核心在 Torvalds 的帶領下已逐漸成熟穩定，而且可以順利執行所有的 GNU 系統工具。故 GNU 團隊便順應 Linux 核心的特性，改寫了他們的 glibc，使其可以適用於 Hurd 核心與 Linux 核心。如此，在這兩個平台上就有了一致的程式開發環境，使得所有的 GNU 程式可以在這兩個平台之間順利移植。

比起過去的 libc5， glibc 系列 (一般又稱之爲 libc6) 除了有完整的國際化與本土化支援外，同時還符合許多標準與規格，使得在 glibc 下開

發的程式可以很容易移植到其他 UNIX 平台去。這些標準包括 :

1. ISO C:

   ISO C 是 International Standard for the C programming language 的縮寫，此標準明定了 C 語言的語法，標準 C 函式庫應具備那些標頭檔、巨集定義、函式與物件 .... 等等，幾乎在任何平台上的 C 語言 (包括非 UNIX 平台) 都支援此標準。

2. POSIX:

   POSIX 是 Portable Operating System Interface for Computer Environments 的縮寫，它是 ISO C 的延伸，明定了一個可移植的作業系統所應具備種種條件，其範圍不只有系統函式庫而已，還同時包括一些標準的工具程式、系統核心應有的特色與實作、以及在 C 函式庫中某些與作業系統相關的低階控制支援 (如系統呼叫窗口) 等等。由於 glibc 是完全按照 POSIX 的標準來實作的，同時搭配了符合 POSIX 標準的 Linux 核心，故在此環境下開發的程式可以做到完全符合 POSIX 的規格。

3. Berkeley Unix:

   Berkeley Unix 泛稱柏克萊大學所開發的 UNIX 系列作業系統，包括 4.2 BSD、 4.3 BSD、 4.4 BSD 以及早期的 SunOS。這些系統的 C 函式庫中有許多傑出的設計，但卻沒有在上述兩個標準中，包括 select() 函式、 sockets .... 等等，這些在 glibc 中都有支援。

4. SVID:

   SVID 是 System V Interface Description 的縮寫，它是一份描述 AT&T UNIX System V 系統規格的文件，它是 POSIX 標準的延伸。 Glibc 實作了大部分的 SVID 規格要求，其中較重要的包括了行程之間的通訊標準以及分享式記憶體 (shared memory)，至於其

他的部分則較不常使用。實作 SVID 主要的目的是希望可以做到與 UNIX System V 的相容與程式的可移植性。

5. XPG:
   XPG 是 X/Open Portability Guide 的縮寫，是由 X/Open Company, Ltd. 所發表，同時 X/Open 還擁有 UNIX 商標的版權。而這份規格不但是 POSIX 標準的擴充，同時也明定了一個 UNIX 作業系統所應符合的要求。其中包括了 iconv() 字集轉換介面，以及部分 BSD 與 SVID 的特色。

除了上述的規格外， glibc 還內含了 GNU 特有的特色，稱之爲 GNU Extension。這些特色在某些情況下可以方便程式的撰寫與維護，但就不見得可以移植到其他 UNIX 平台上，故在可移植性的考量下使用時必須留意。

7.3.3 GLIBC 的內容

由於 glibc 囊括了幾乎所有的 UNIX 通行的標準，可以想見其內容包羅萬有。而就像其他的 UNIX 系統一樣，其內含的檔案群分散於系統的樹狀目錄結構中，像一個支架一般撐起整個作業系統。以 glibc-2.2 爲例，這些檔案群主要包括 :

1. 分享函式庫群 :
   這是 glibc 的主體，分佈於 /lib 與 /usr/lib 中，包括 libc 標準 C 函式庫、 libm 數學函式庫、 libcrypt 加密與編碼函式庫、 libdb 資料庫函式庫、 libpthread 行程多執行緒函式庫、 libnss 網路服務函式庫 .... 等等。這些都是可分享函式庫，檔名都以 .so 做結尾。

其中， /lib/ld*.so 是程式與函式庫連結的工具。有的用於程式編譯時將程式與函式庫內的函式物件連結，在只支援靜態連結的系統中，此連結方式就是直接將所需的物件自函式庫中抽出來與程式的可執行檔相連，而在支援可分享函式庫的系統中，在程式編譯時期的連結只是在執行檔中紀錄了那些函式物件是存在那個函式庫檔案中，等該程式開始執行時，則由另一個負責動態連結的 ld*.so 將所需的函式庫連結好並執行。

一般而言，負責程式編譯時期的連結器檔名爲 ld.so，而負責程式執行時的動態連結器檔名爲 ld-⟨*version*⟩.so 或 ld-linux.so (在 GNU/Linux 系統中)。

2. 函式庫標頭檔與程式開發元件 :

   這些標頭檔檔名都以 .h 爲結尾，全部在 /usr/include/ 底下，其內容爲函式庫中各函式的宣告、巨集定義、資料物件的型別 .... 等等，這些都是程式開發者不可或缺的部分。

   除此之外，在 /usr/lib/ 中還有若干 .o 與 .a 的檔案，這些是程式編譯過程中要連結爲可執行檔時所需的元件，有些則爲上述可分享函式庫的靜態連接版本，而後者可以在某些特殊場合下需要靜態連結程式時使用。

3. 函式庫說明文件 :

   在一般的 UNIX 系統下，這些說明文件是放在 /usr/man 或 /usr/share/man 底下，統稱爲 man pages，其底下還分若干章節，其中第二章 (man2) 講的是系統呼叫，而第三章 (man3) 講的就是 libc 標準函式庫，這些都是系統開發者重要的參考資料。

   而在 GNU 的系統中，除了 man pages 之外，還有一套稱爲 info 的文件資料系統，而且裡頭的說明往往比 man pages 還要詳盡，這在

glibc 中也不例外。glibc 的 info 文件位於 /usr/share/info/libc.info*，本文中有許多素材就是取自這些文件的內容。

4. 字集轉換模組與區域化資料庫：

   這些是與程式國際化與本土化相關的部分，主要可分成四大塊：/usr/lib/gconv/ 內含大量的字集轉換模組，大部分是各種字集及編碼方式與系統的基底字集之間的 轉換。第二塊是 /usr/lib/locale，內含以系統基底字集寫成的區域化資料庫 (locale)，像是 LC_CTYPE、LC_TIME .... 等等。第三塊是 /usr/share/locale/，內含可跨平台使用的區域化資料，主要是各應用程式的訊息翻譯部分。而最後一塊是 /usr/share/i18n/，其內容是各區域化資料庫的原始碼，以及系統支援的內碼對應表 .... 等等。

5. 時區資料庫：

   主要分佈於 /usr/share/zoneinfo 底下，內含世界各地時區與格林威治時間的轉換資料。

6. 其他工具程式與設定檔：

   工具程式分佈在 /usr/bin 與 /sbin 底下，包括一些轉碼與區域化資料庫相關的程式如 iconv, locale, localedef 等，以及用來顯示應用程式與可分享函式庫相依關係的 ldd, 還有可分享函式庫搜尋路徑管理程式 ldconfig .... 等。而其相關的設定檔則位於 /etc 底下。

7.3.4 GLIBC 的函式物件

從程式開發者的角度來看，glibc 內含的函式物件可分爲底下幾大類，它們分別依據上述的各種規格來實作的，同時系統呼叫的窗口也分佈在這些函式物件當中：

1. 錯誤處理常式：包括函式錯誤偵測、錯誤訊號編碼 (errno) 與錯誤訊息顯示等。

2. 記憶體管理與陣列：包括動態記憶體配置、字串處理、陣列資料處理等。

3. 程式國際化與區域化：包括區域化資料庫 (locale) 的呼叫介面、字集編碼轉換、訊息翻譯的顯示、字元處理與分類等。

4. 輸入與輸出：包括標準輸出入 (standard I/O)、檔案處理、低階輸出入系統呼叫等。

5. 行程管理與行程通訊：包括子行程的產生、群組管理等函式群，而通訊方面則包括管道 (pipe, fifo)、 sockets、分享記憶體、行程間訊號處理 .... 等等。

6. 系統服務：包括取得作業系統資訊、可用資源分配管理、系統訊息輸出 (syslog)、使用者群組與權限管理、各種網路服務如 DNS、NIS .... 等等。

7. 其他函式元件：包括各種搜尋與排序演算法、字串符號搜尋與正規化表示式、時間表示、數學函式群、字串加密演算法、資料庫系統、行程多執行緒支援 .... 等等。

參考資料：

1. glibc man pages, info libc[115].

2. ELF 規格書[116, 117]。

3. ISC 與 POSIX[118]。

4. XPG 規格書[119]。

7.3.5 C++ 函式庫

GNU C++ Library[120] 。

7.4 多媒體

- 編碼 /解碼 /壓縮 :
  - GSM 06.10 lossy speech compression[121].
- 影音 :
  - SDL - Simple DirectMedia Layer[122].
  - SMPEG - SDL MPEG Player Library[123].
  - libogg[124].
  - libvorbis[125].
  - audiofile[126].
  - libsndfile[127].
- 圖形
  - ImageMagick[128].
  - libpng - Library for Portable Network Graphics[129].
  - libmng - Library for Multiple-image Network Graphics[130].
  - libtiff - Library for Tag Image File Format[131].
  - libungif - A library for using uncompressed GIFs[132].
  - xpm - XPM pixmap library for the X Window System.
  - gd - A graphics library for drawing .gif files

  - imlib - An image loading and rendering library for X11R6
  - libjpeg - A library for manipulating JPEG image format files
  - netpbm
  - svgalib
- 3D
  - Mesa[133].

7.5 通訊

- OpenH323 - H.323 protocol stack[134].
- Vovida - MGCP, SIP, RTP[135].
- OpenSS7 - Signalling System Number 7 protocol stack[136].

7.6 系統安全

- OpenSSL[72].
- libsafe - Protecting Critical Elements of Stacks[137].
- Cyrus SASL Library - Simple Authentication and Security Layer[138].

7.7 圖形介面：X 函式庫

7.7.1 Xlib 函式庫簡介

X Window 系統是 UNIX 世界中標準的圖形操作介面，它是在 1984 年由麻省理工資訊科學系與迪吉多公司合作開發的一個專案計畫，其目是要

發展一個兼具可移植性與網路通透性的視窗系統。該專案計畫首度發表的是第十版的視窗系統 (X10)，到了 1987 年發表了第十一版 (X11)，歷經多年的開發演進，到了 1994 年發表了 X11R6，也就是目前 UNIX 系統上通行的版本。而原本負責此專案計畫的成員，也在 1996 底組成了一個稱爲 X Consortium 的組織，持續 X11R6 的開發與維護，而其最新的改進版是去年才問世的 X11R6.5。

由於 X Window 專案計畫的出現，使得各 UNIX 廠商與軟體開發組織可以以 X Consortium 所發展的程式碼爲藍本，在他們的系統上建立圖形化視窗環境。爲了達到完整的可移植性與網路通透性，各廠商與軟體開發組織所實作出來的 X Window 版本都遵守相同的協定，即 X 協定，同時也採用同樣的函式庫呼叫介面。因此，只要是 X Window 環境下開發出來的程式，原則上都可以不需修改而移植到其他平台的 X Window 環境 (只要它的 X Window 版本與原開發環境的版本相同或更早即可) 編譯執行。

而在 GNU/Linux、 FreeBSD 等 x86 平台的自由 UNIX 作業系統中，其 X Window 系統是來自 XFree86 計畫，顧名思議，該計畫的主要目的是提供一個免費、自由的 X Window 系統給 x86 個人電腦上的 UNIX 使用。而原先 GNU/Linux、 FreeBSD 等是在 x86 電腦下開發的，但近年來它們已逐漸移植到 DEC Alpha, Sparc .... 等高階的電腦系統上去，因此 XFree86 也順應做了移植，可以在那些非 x86 的硬體系統上執行了。該計畫也是依照 X Consortium 的標準而開發的，例如這兩三年來通行的 XFree86-3.3.X 系列，主要就是以 X11R6.3 爲藍本。

但到了近一兩年來情況稍有改變，由於 GNU/Linux 平台上的多國語言與 Unicode 支援的需求日益迫切，同時爲了順應近年來新的軟應體技術，使得傳統的 X11R6 實作方式已漸漸不縛使用。然而 X Consortium 在這些部分的開發腳步卻不夠快，或甚至仍未起步，故到了 XFree86-

4.0.X 系列，除了遵循 X11R6.4 的標準以外，同時更進一步地開發了 Unicode 的支援、 TrueType 字型支援、模組化顯示卡驅動程式、高速繪圖介面、並逐步修正過去 X Consortium 所發表的程式碼中的錯誤 .... 等。可以這麼說， XFree86-4.0.X 系列比起過去的版本有著相當長足的進步。而這些改進，未來也許會逐步出現在 X Consortium 新版的 X11R6 中。

X Window 系統採取的是 Server/Client 的模型而運作。所謂的 Server 指的是 X Server, 它通常掌管一個完整的 Display。根據定義，一個傳統的 Display 包含一個顯示器、一個鍵盤、及一個滑鼠，或者還有其他選擇性的輸出入裝置，換句話說，它就是一個完整的圖型桌面裝置。而 Client 指的是在此 Display 中執行的所有 X Window 應用程式，它們需要在螢幕上繪圖、需要接收滑鼠、鍵盤等輸入 .... 等，都必須向 X Server 發出請求，由 X Server 代爲完成。而 X Server 與 Client 之間的溝通協定就稱之爲 X 協定。此 X 協定不僅可用於本機的 Display (亦即 X Server 與 Client 都在同一部機器上執行)，它更具備了網路的通透性，也就是 X Server 與 Client 可以在不同機器上執行，例如將遠端的 Client 視窗顯示到本地的 X Server 上，而本地的使用者在使用時就和操作本機執行的 Client 一樣，不會有分別。此網路的通透性正是 X Window 系統最強大的特點之一。

由 X Server 掌控的 Display 只是圖形介面的底層平台而已，它還不是我們平常所接觸到的桌面環境。我們所用的桌面環境是由「桌面管理程式 (Window Manager)」所提供，它擁有方便的功能表選單、啓動應用程式的按扭圖示、可以幫我們放漂亮的底圖、及管理桌面上所有的視窗 .... 等等的功能，事實上，「桌面管理程式」在 X Server 的眼中，也不過是一個 X Client 而已，只是它的功能比較特殊，專門負責管理整個工作桌面。由於 X Window 系統並不將桌面管理程式內建在 X Server 或

Display 中，因而給我們一個彈性：我們可以依自己的喜好挑選桌面管理程式來用。這類的桌面管理程式可以說形形色色，有功能小巧陽春的 twm 與 fvwm、有長得很像 Windows 95 介面的 fvwm95、有相當眩麗的 enlightment、而有的功能甚至超越了單純的桌面管理程式，還同時提供了一組程式庫與應用程式、管理工具等等，讓我們的桌面更加好用，如 KDE 或 Gnome 等。

而這些形形色色的 X 應用程式 (Client) 與 X Server 之間的溝通管道，就是實作並傳輸 X 協定的 X Window 底層函式庫：Xlib。 Xlib 在 X Window 中的地位，就好像 libc 在整個作業系統中的地位一樣。若要做進一步的類比， X Server 的角色就如同作業系統的核心， libc 提供了系統核心系統呼叫的窗口，同樣的 Xlib 可以直接與 X Server 連繫傳送訊息。若要撰寫 X Window 的應用程式，則最底層可用的函式庫就是 Xlib。

7.7.2 Xlib 函式庫概觀

在 X Window 的世界裡，可以說所有的動作都是由「事件 (Event) 」所觸發並完成的，不論是對 X Client 或是 X Server 都是一樣。從 X Client 的角度來看，每個 X 應用程式內部都有一個處理事件的迴圈 (event loop)，程式靜靜地等待事件的發生，一旦 Xlib 截獲一個屬於該應用程式的事件並傳送給它時，該事件就會在事件處理迴圈中產生相應的動作，處理完後，又會回到原點，等待下一個事件的發生。可能發生的事件有很多種，像是其他的視窗傳來訊息、鍵盤滑鼠有了動作、桌面管理程式要求改變視窗的大小狀態 .... 等等。

同樣的，在 X Server 這邊也是等待事件發生，以提供適當的服務。它同時還監控著整個 Display 的所有裝置，如果有任何訊號輸入 (如來自鍵盤或滑鼠的輸入) 時，它會將該訊號打包成「事件」的包裹，經由

Xlib 傳送給等待接收訊號的 Client (即前景視窗)。由於 X Window 具備網路的通透性，故在設計時特別考慮到網路傳輸效率的問題。如果網路速度不夠快時，每次 Client 一有服務請求就要求 X Server 馬上回應，顯然太缺乏效率了。因此在 Xlib 中有一個事件的緩衝區，當 Client 提出一連串的服務請求時，這些請求並沒有馬上送往 X Server，而是暫時保留在緩衝區中，等到必須要送出的時間點時，才會一口氣送出，而讓 X Server 將這一連串的服務請求一口氣處理完畢。通常這個時間點是發生在 Client 送完了一連串的請求後，停下來等待下一個屬於它的事件時，或者 Client 呼叫了 Xlib 中特定的函式要求 Xlib 馬上將它的請求發送給 X Server。

此種以「事件處理」爲基礎的運作模式，使得 X Window 的程式寫起來，與一般的程式有些不一樣。同時一個視窗的程式由於牽涉到許多可調整的細節，像是視窗大小、顏色、底圖樣式、線條粗細、字型 .... 等等，因而使得程式內部會用到的資料結構與函式呼叫往往會很複雜。但大體而言，一個標準的 X Window 程式其內部的基本執行程序大致如下：

1. 呼叫函式 XOpenDisplay() 與 X Server 取得連繫。

2. 初始化自己的視窗，包括一切屬性設定，同時要將一些視窗屬性的「提示」告訴桌面管理程式。注意這裡只是「提示」而已，因爲桌面管理程式管理所有的視窗，在某些情況下它可能無法完全按照應用程式的要求來執行 (例如應用程式要求了超過桌面範圍的視窗位置) ，故它多少會自行調整一下。但大體而言它會儘可能按照應用程式的「提示」做到。

3. 設定應用程式視窗欲接收的「事件」種類，依應用程式的不同，其希望接收的事件也不同，若程式不需要的事件就不用接收，如此就

可以省下不必要的效率浪費。

4. 進入事件迴圈，等待事件，處理事件，直到程式結束。

程式大部分的工作都是在事件迴圈中運作，包括重畫視窗本身，因爲自己的視窗可能會被別的視窗遮住，當別的視窗移開時，桌面管理程式就會送一個「要求視窗重畫」的事件過來。因此， Xlib 內部的函式中，除了用在應用程式初始化與其他特殊用途以外，其餘幾乎都是用在處理事件迴圈中。這些函式群大至可分類如下 :

1. 與 X Server 連繫、取得 X Server 內部可用資源與相關資訊等。
2. 視窗的產生、屬性設定、與桌面管理程式溝通及發送「提示」等。
3. 繪圖函式群與繪圖屬性設定。其中繪圖屬性指的是 Graphics Context (GC)，它是一個大型的資料結構，可用來指定線條粗細、字型與點數、顏色、底圖 .... 等等。
4. 顏色處理與色板 (即可用的顏色數與顏色種類) 設定。
5. 事件種類與各事件的處理方式，包括取得鍵盤的輸入與滑鼠的動作等。
6. 程式國際化。在舊版 X11R5 的時候，已開始有程式國際化的雛型，但在某些方面還不夠完整，直到 X11R6 才接近成熟。這裡的程式國際化包括 FontSet 的概念、 X Locale 的設定、國際化文字輸出 (繪圖)、輸入 (輸入法程式與 XIM 協定)、複合字串 (Compond Text) 的處理 .... 等等。而在 XFree86-4.0.X 系列，還多了一組 Unicode/UTF-8 的處理函式，內容相當豐富。
7. 各視窗應用程式之間的通訊機制。主要有三種，由簡單到複雜依序爲「訊息事件」的發送、使用視窗的 property 通訊、以及使用

Selection 機制通訊。越複雜者所能通訊的資料量越大、能處理的資料型態可以越多樣、同時也越可靠。

8. 資源管理與使用者設定。前面我們已說過一個 X Window 的程式可調整的部分相當多，而這些可調的部分可以以一個特別的格式存在特定的「資源檔」中，如使用者 HOME 目錄下的 `.Xdefaults` 或 `.Xresources` 檔中。因此，Xlib 提供了一組特別的函式用來取得這些資源檔的資料。

9. 桌面管理程式專用的函式群與相當協定，這些函式賦與程式管理桌面與其他應用程式視窗的能力。

7.7.3 各式各樣的 Tool Kit

由上可知，使用 Xlib 直接來開發應用程式可能不是件容易的事，因爲中間牽涉到許多視窗與圖形介面的低階操作，過程很複雜，故程式寫起來會很長。爲了方便 X Window 的程式開發，因而有各式各樣的 Tool Kit 出現。這些 Tool Kit 多半是以函式庫的形式存在，但有些是編寫成直譯語言 (script) 的形式，在使用上更爲方便。不論是何種形式，它們都是以 Xlib 爲基礎建構起來，它們將每個 X Window 程式都必須的視窗初始化工作、事件迴圈與事件處理、各視窗間的通訊、長用的圖形介面元件 .... 等等冗長的程式碼，包裝成幾個簡單的函式，可以讓程式開發者直接使用。故使用 Tool Kit 可以讓程式寫起來較爲乾淨簡潔，有助於提升程式開發的質量。

在這些 Tool Kit 中，特別引進了在 Xlib 中沒有特別強調的概念，稱之爲 Widget Set。簡而言之，Widget Set 可以視爲一個圖形介面的「外觀」或「樣式」，它可以很簡單只是一兩個圖形元件，如一個簡單視窗，或視窗中某個按扭，也可以複雜到形成一個具備特殊功能的視

窗，如接受文字輸入的視窗，還可以將文字 Cut/Copy/Paste 到別的視窗上 .... 等。所有的 Tool Kit 中幾乎都內建了許多 Widget Set，而這些 Widget Set 的外觀、功能在不同的 Tool Kit 間或多或少會有差異。因此，使用不同的 Tool Kit 設計出來的程式，可以讓人一眼看出來其風格的不同。

目前常見的 Tool Kit 有許多種，大至分類如下 :

1. libXt, libXaw: 這是 X11R6 系統中內建的 Tool Kit，讓我們除了使用 Xlib 之餘也可以試著用它們來撰寫程式。大部分的 X11R6 內附的工具程式或應用程式都是用這兩個 Tool Kit 寫成的。另外，還有兩種稱爲 Xaw3D 與 Xaw95，可以讓 libXaw 的程式外披上 3D 立體效果或像微軟 Windows 95 視窗的外衣。

2. Motif/Lesstif: Motif 可以說是過去在 X11R5 時代最強大的一個 Tool Kit，同時也是當時業界採行的標準。幾乎絕大部分在當年開發的商業軟體都是用 Motif 寫成的。由於 Motif 是商業軟體，故在某些場合下無法使用 (最近它的授權限制較爲放寬了)，爲了讓自由軟體世界也可以使用以 Motif 爲基礎的軟體，因而出現了 Lesstif。它可以說是模仿 Motif 的介面重新實作的 Tool Kit，以自由軟體的授權散佈，但目前它只完成到相當於 Motif-1.2 版的程度而已，而 Motif-2.X 版的相容與支援目前仍在開發中。

3. Qt/KDE: Qt 是一個用 C++ 物件導向式語言完成的 Tool Kit (上文的 Tool Kit，包括 Xlib 都是 C 語言的介面)。它是由 Trolltech 公司所發展的，它不僅是繼 Motif 之後功能最齊全且完備的 Tool Kit，同時使用上比 Motif 容易，而且還跨平台 : 可以同時跨 UNIX 與微軟視窗系統的平台。因此，使用 QT 開發的程式，理論上可以在不需修改 (只圖形介面部分的程式碼) 而可以直接在 UNIX 與微軟視

窗系統上編譯使用。

KDE 是以 Qt 爲基礎而完成的桌面環境系統，它是目前自由軟體世界使用最廣泛的桌面環境系統之一。它不僅是桌面管理程式而已，同時還有一組完整的應用程式、工具程式、與函式庫、程式開發工具。使用它的函式庫，可以讓應用程式與整個 KDE 系統整合在一起，而享有 KDE 環境所有的資源與服務，當然，也繼承了 Qt 所有的特性。

由於 KDE 是以 GPL 的自由軟體，而 Qt 是商業公司發展的軟體，二者在軟體授權上不相容，故曾經有一度引發自由軟體世界的困擾，而無法完全普及使用。直到近一兩年來， Trolltech 公司終於同意讓某些 Qt 版本以 GPL 授權散佈，局面才豁然開朗。 Qt 的 GPL 化，可以說是自由軟體世界相當歡欣鼓舞的一刻。

4. gtk+/Gnome: gtk+ 與 Gnome 的組合與 Qt/KDE 的組合很像，也是企圖開發出一套功能強大的 Tool Kit 與桌面環境系統，它們的開發者是 GNU。事實上，它們的開發正是源自於早年 Qt 不是自由軟體的困擾。儘管現在困擾已消失了，但這一套系統仍繼續往前走，同時 Qt 與 gtk+ 兩大陣營也開使合作，共同開發二者可互通或相容的程式發展介面。

   gtk+ 原本並沒有要發展成完整的 Tool Kit 的企圖，剛開始時只是 GNU 的一個圖型影像處理程式 Gimp 背後的一個函式庫而已，到了 Gnome 開始發展時才逐漸往完整的 Tool Kit 方向發展。它最大的特色是可以和各式各樣的程式語言結合使用，包括 C/C++、 Ada、 Perl、 Python, 而其他像 JavaScript、 Guile、 Objective-C、 Pascal .... 等等也在開發中，故使用上彈性相當大。

5. Tcl/Tk: Tcl 是一種直譯式 (script) 的語言，我們只需寫幾個簡單的指令，透過 Tcl 直譯器就可執行許多複雜的工作，而 Tk 則是以 Tcl

語言爲基礎完成的 Tool Kit，它可能是撰寫 X Window 應用程式中最方便使用的 Tool Kit。

除了以上這些常見的 Tool Kit 以外，還有許多足繁不及備載，像是 libforms、 FLTK 或 Java/JavaScript 之類等等。

參考資料：

1. X Consortium[139].

2. XFree86 Project[140].

3. Xlib Programming

   a) Xlib - C Language X Interface, by X Consortium Standard[141].

   b) The Definitive Guides to the X Window System Volume 1, Xlib Programming Manual (for Version 11) Author: Adrian Nye Publisher: O'Reilly

   c) The Definitive Guides to the X Window System Volume 2, Xlib Reference Manual (for Version 11) Author: Adrian Nye Publisher: O'Reilly

   d) The Definitive Guides to the X Window System R6 Update for the R5 Editions of vols. 1, 2, 4, & 5 Programmer's Supplement for Release 6 of the X Window System. Author: Adrian Nye Publisher: O'Reilly

4. libXt: The Definitive Guides to the X Window System Volume 4, X Toolkit Intrinsics Programming Manual Author: Adrian Nye and Tim O'Reilly. Publisher: O'Reilly

5. Motif[142, 143]

6. Lesstif [144]
7. Qt[44], KDE[145].
8. Gtk+[146], Gnome[147].
9. FLTK - Fast Light Tool Kit[148].
10. Tcl/Tk[101, 102]

7.8 語言

- Pango - layout and rendering of internationalized text[149].
- linunicode - A Unicode character manipulation libary.
- frebidi - a Free Implementation of the Unicode BiDi algorithm[150].
- libtabe[151].
- Canna[152].

7.9 跨平台

- wine - an implementation of the Windows APIs on top of X[153].
- pwlib - Portable Windows Libaray[134].

7.10 其他

- db - Berkeley DB[154].
- freetype - TrueType font rendering engine[155].

- ncurses[156].
- libxml[157].
- bzip2[158].
- slang - a multi-platform programmer's library[159].
- GNU Pth - The GNU Portable Threads[160].
- cracklibs - Password Checking Library[161].
- fnlib - A color font rendering library for X11R6[162].

第八章

作業系統核心與驅動程式

8.1 作業系統核心

作者：陳開基

8.1.1 簡介

一部運作正常的電腦，不論是超級電腦還是桌上型 PC ，大致上主要由三個部份所構成的， 這三個部份分別是硬體，作業系統和應用程式。硬體，顧名思意，就是像 CPU ， 硬碟等這些構成電腦的電子元件，而應用程式則是使用者利用來完成特定目標的程式， 例如 ms word 是專門作文書處理的應用程式， netscape 是 brower 。應用程式是一個集合 ，代表所有執行於作業系統以上的軟體，基本上我們每天開機使用的都是一個又一個的應用 程式。那作業系統又是什麼？ 作業系統位於幕後，是位默默耕耘的功臣，雖然不像硬體看得見，或是像應用程式可以直接 使用，它卻影響整部電腦的效率甚巨。它的兩個最重要的任務是提供應用程式介面及資源管理。

控制硬體是件極複雜的事，以讀寫軟碟爲例，必須啓動馬達， 移動

磁頭等等，如果開發應用程式須要包含硬體的控制，那麼難度馬上增加好幾倍， 而且不同廠牌的硬體又會有不同的操作方法，等於對每一家硬體廠商須要重寫一次。 還好，應用程式控制硬體的方式大同小異，以軟碟爲例，不外乎格式化 (format)，讀出資料，寫入資料等少數幾個動作。 把這些硬體的常用動作，寫成類似函式庫的形式，提供給應用程式呼叫， 就是作業系統核心的主要任務之一。 這些由作業系統核心提供給應用程式用的類函式統稱爲系統呼叫 (system calls)， 從此以後，應用程式不必再擔心軟碟的運作，它只須呼叫 system call 的 read 或 write ，它甚至不須要管它要寫入資料的是硬碟還是軟碟， 使得應用程式的開發可以專心在它的特色功能上。

作業系統依特性可分爲單工與多工， DOS 是一個單工的作業系統，單工作業系統的預設是，一個應用程式執行完畢後才會執行另一個程式，因此它允許應用程式可以無限制地使用系統的資源，例如 CPU 時間，記憶體和 IO 等等，這個應用程式甚至可以佔據系統，不再將主導權交回作業系統。這類單工的作業系統只須提供上述的系統呼叫便已足夠。而多工的作業系統，如 Linux ，它允許多個應用程式同時進行，共用 CPU 時間和記憶體，因此只提供系統呼叫的服務是不夠的，還必須做資源的管理，讓這些同時在執行的應用程式不會互相干擾，例如一個程式不能寫入另一程式的記憶體，才能確保這些應用程式都能順利地完成任務。

提供應用程式介面及資源管理並不是完全獨立的，而是相輔相成，因爲當程式透過系統呼叫去存取 IO 或要求記憶體時，系統呼叫會呼叫作業系統，作業系統將所有來自程式的要求整合後，在滿足它們的要求時，也作資源管理。

一般多工作業系統可分爲下列次系統：行程管理 (爲了簡單化，此處將行程和程式視爲同義)，記憶體管理，檔案系統， IO 及網路系統。這

些次系統分別管理整部電腦的一些重要資源，行程管理是管理 CPU時間的分配，其它的次系統從名字，即可得知它們所管理的資源。

8.1.2 Unix 歷史簡介

現行主要的開放式作業系統有 Linux， FreeBSD 等等，都是以 Unix 爲基礎。所以我們先介紹一下 Unix 的發展過程。

1969 年 Ken Thompson 在 AT&T的 Bell Lab，利用一部閒置的 PDP-7 研究新的作業系統，這是 Unix 的前身。之後不久， Dennis Ritchie 加入 Ken Thompson 一同開發這個新作業系統，由於 Dennis Ritchie 曾參與 Multics 計劃，因此 Unix 的雛型和 Multics 非常相似，但他們也加入了 MIT 的 CTSS 及 XDS-940 等作業系統的一些概念。

他們繼續開發 Unix，但 Ken Thompson 和 Ritchie 不喜歡組合語言，因而設計新的程式語言作爲開發作業系統的工具，這個語言後來獲得全球程式設計師的熱愛，那就是 C 語言。 Unix 第三版是用 C 語言寫成的， C 語言具有比組合語言來得容易開發與維護等優點外，它也大幅增加硬體的可移植性，奠定了 Unix 普及化的基礎。

在這之前， Unix 只在是存在於 AT&T 內部， 1976 年他們首次將 Unix 流傳到 AT&T 以外的地方，此時 Unix已經發展到第六版（版本是根據當時出版的手冊定的）。

後來 Unix V7.0 又被移植到 VAX 上，稱爲 32V 。由於 Unix 輕薄短小，許多人加入 Unix 的研發工作，如 University of Illinois, Havard, DEC 等等，其中以 UC-Berkely 的成果影響最深遠， UC-Berkeley 的 Bill Joy 和 Ozalp Baboghe 以 32V 爲基礎，加入虛擬記憶體，要求分頁，分頁替換等重要功能後，發表爲 3 BSD Unix， BSD爲 Berkeley Software Distribution的 縮寫。因爲它具有大型的記憶體管理機制，因而

獲 DARPA 的支持，繼續發展爲 4.x BSD，到了 4.2 BSD 時，它已支援 LAN(Ethernet及 Token Ring)及 WAN 。

在這段期間中， AT&T 也持續地爲它的 Unix 增加新功能， 1978 年 AT&T 成立 USG(Unix Support Group)，商業化 Unix，它發表的第一版本是 System III，下一個版爲 System IV。

從此以後， Unix 就以 System IV 和 4.x BSD 這兩大主流蓬勃發展，到了八十年代末才開始有一些整合的動作，如 Sun Solaris Os 及 POSIX。

我們不得不問， Unix 到底有什麼特別，能使它獲得空前的成功，它和之前作業系統主要的差別在於二點：

一， 之前的作業系統都是龐然大物，幾乎都是電腦內執行的最大程式，而且都是針對特定硬體設計的。相反的， Unix 的設計哲學是小而美，目標是能在任何的小系統上執行。

二， Unix 的哲學是作業系統核心只提供少數不可或缺的功能，其它的功能則以使用者程式的型式加上去。

這兩點使 Unix 成爲革命性的作業系統。

8.1.3 Linux

簡介

由於版權問題， Unix 的源碼不再適用於教學， 1987 年 Andrew Tanenbaum 遂寫了 MINIX 作爲教學的工具， MINIX 的意思爲 mini-Unix，它是一個簡化的作業系統，適合入門者學習作業系統，因爲簡單，剛開始時獲得眾人的青睞，但好景不長，原因是它過於簡單反而不切實用。

1991 年， Linus Torvalds 開始使用 MINIX，他對 MINIX 提供的功

能不滿意，自行發展 Linux，他把 Linux 的原始碼放在 Internet 上，允許人們自由使用 (under GNU Public License)。 Linux 是第一個完全免費的 Unix，很快地，許多人開始修改及加強 Linux，如今， Linux 除了可以在原先設計的 Intel x86(x >=3) 上執行外，它也被移植到 Alpha, Sun Sparc, Motorola 68K, MIPS, PowerPC 等等的平台上。

由於 Linux 提供和 Unix 類似的介面，因此，在資源的抽象化上及資源分享的型式上就必須和 Unix 相同，它和其它 Unix不同的地方在於實作這些介面時所使用的資料結構及演算法。

Linux 是一個多工的作業系統，許多程式可以一次讀到記憶體內，作業系統執行某個程式一陣時間後，就會切換去執行另一個程式。所以記憶體是被空間分割，分成數個區塊，每個區塊稱爲 memory partition，內含一個隨時可以執行的程式。 CPU 是被時間分割，每段時間執行一個在記憶區塊內的程式，時間到了便切換到另支一程式。在 Intel x86 平台上， CPU 每秒約切換 100次，在 Alpha 平台則切換約 1024 次，這些切換次數可依需要而修改，大致而言，降低切換次數會使諸如鍵盤，mouse 的反應變慢，但應用程式執行的效率會提高，相反地，如果提高切換次數則會使程式的執行變慢，因爲更多的 CPU 時間被用來切換程式。

Linux 核心和傳統的 Unix 一樣，是屬於集成式的作業系統核心 (monolithic kernel)，和目前流行的微核心 (microkernel) 不同。它們將行程管理，記憶體管理和檔案系統包在一起，成爲一個單一的可執行檔，而週邊硬體裝置管理則另外分開，成爲一組驅動程式，每一個驅動程式的目的是控制某一類型的硬體裝置，例如控制軟碟機。這種設計是爲了降低核心更動的頻率，不必爲了新硬體裝置更改核心，而且驅動程式也比較好寫。但是 Linux 核心的進步非常迅速，這種設計反而不利於核心的實驗更新，爲了克服這個缺點， Linux 提出模組 (module) 機制，這是

種軟體容器，它和核心的介面要比傳統 Unix 的驅動程式來的有彈性，可以用來提供新功能給核心，當然也適用於寫驅動程式。

目前 Linux最新的核心版本爲 2.4.8，每個版本有三個數字，第一個數字是核心有重要進 展時才會增加，例如當初從 1.x.x進 到 2.0.x主要歸功於 SMP支援 (簡而言之，就是支援多 CPU的平台)。第二個數字從 1.0版後 就有特殊的意義，奇數代表發展版 本，目的在於嚐試新功能，偶數是穩定版本，目的主要在除錯，而且供人使用。例如 2. 1.x就是發展版本，重點在增加新功 能，當新功能累積到某一程度後，就會升級成 2.2.0，從此之後重點會擺在除錯，然後 再從 2.3.0去試新功能，所以 Linux通常 會有兩個最新版本，如 2.2.x和 2.3.x。第三個數字是小改進的時候就可以增加，例如修 掉幾個臭蟲等等。

以下就行程管理，記體管理，檔案系統及週邊裝置管理等方面做介紹

行程管理

在介紹行程管理之前，我們先提一下 CPU 的執行模式，基本上，許多 CPU 除了執行指令外，還會訂定數種優先等級，當 CPU 在不同的優先等級下執行相同的指令會得到不同的結果。 Intel x86 的保護模式定了四種等級，但是 Linux 只用到其中兩種，核心模式 (kernel mode) 及使用者模式 (user mode)。核心模式有較大的權限，可以任意存取週邊設備，通常作業系統核心都是以核心模式在執行，而使用者模式權限小，限制多，應用程式均以使用者模式執行。

當應用程式正常執行時，有下列幾個狀況會切換到核心模式 (事實上就是 CPU 從正在執行的程式切換到作業系統核心碼)，一是程式主動呼叫系統服務函式，例如程式想讀取硬碟裡的檔案，或是程式向系統要求

多一點記憶體。二是用硬體要求 CPU 反應，例如你在鍵盤敲入一個字母，負責鍵盤的晶片要求處理這一筆資料，另一例是電腦內的時鐘每一固定時間會通知 CPU，告訴 CPU 說時間又過了一個 tick(在 x86上，約 0.01秒)， CPU 就會切換到核心模式，執行與此事件相對應的核心碼，這時有許多要做，其中一項是計算目前這個程式已經執多久了，是否該換手執行讓下一個程式了，如果時間未到，核心事情處理完後會讓原先的程式繼續進行。幾乎每一種可能發生的事件，核心都早已備好對應的程式碼。三是程式作了不該作的事，例如讀取不屬於它的記憶體，或直接存取週邊設備，這時 CPU 也會轉到核心模式，讓作業系統核心決定如何處置這個有問題的程式，最普遍的處置就是停止這個程式的執行，把錯誤回報到終端機。

瞭解核心模式和使用者模式後，我們可以介紹行程管理了，之前我們並沒有對行程和程式作區分，在這一節必須作區分。例如執行一個從一數到十的程式，每秒數一下，須十秒的時間程式才能執行完畢，我在十一點時執行這個程式，在十一點過五秒又再執行一次這個程式 (幸好 Linux 是多工作系統)，當我執行這個程式時，作業系統會把對應的檔案讀進記憶體，作一些設定，然後讓 CPU 去執行它，第二次執行時作業系統會做同樣的事，但是當然會放在不同位址的記憶區塊 (作業系統不會管這兩個程式是否同一個)。所以到了十一點過八秒時，和這個例子相關的有三個東西，第一是仍然躺在硬碟裡的程式碼，第二是數到八，在記憶中未執行完畢的那段與程式對應記憶區塊，第三是數到三的另一記憶區塊。爲了作區分，躺在硬碟裡仍然稱爲程式，在記憶體中執行的每一個體稱爲行程，這裡有兩個行程，每個行程都是獨立的個體，有不同的變數值，可能也會有不同的狀態，例如第一個行程要到記憶體，第二個行程要不到等等。

作業系統核心中的行程管理主要目的就是管理這些正在執行或等待

執行的行程們 (躺在硬碟裡的程式歸檔案系統管)，行程管理的核心是排程器，目的是爲行程們安排 CPU 的時間。在 Linux 中行程也分等級，共三個等級 SCHED_FIFO, SCHED_RR 和 SCHED_OTHER。前兩種等級有優先權，設計給即時應用軟體使用的，當沒有前兩個等級的行程時，第三等級的行程才有機會執行，這個等級的行程是以類似輪流的方式執行，事實上它們也是互相比等級，等級最高的會選爲下一個執行的對象，但是高等級執行完後等級會重算，通常等級會降下來，所以會輪到當初次高等級的行程執行。等級的算法有一個簡單的公式，在函式 goodness() 中，它位於 /usr/src/Linux/kernel/sched.c 中，改這個公式是 hacker 喜歡的把戲，可以試試看系統如何變化。

記憶體管理

記憶體管理的主要任務有以下幾點：

1. 提供記憶區塊以滿足程式的要求。
2. 保護程式的記憶區塊，確保不受其它程式不當的侵犯。
3. 提供一個機制，讓合作的程式可以共享記憶區塊。

Linux 記憶體管理的基礎是要求分頁的虛擬記憶體模型 (demand paged virtual memory model)，讓我們就虛擬記憶體，分頁和要求分頁一一作說明：

虛擬記憶體是說程式直接存取的記憶體不再是實體記憶體的位址，以 x86 爲例，在 DOS下的程式，它可以自由地在某一記憶體位址中填入任何一個值，例如在 1000:0000 這個 byte上填入 ff。但是在多工的系統中這麼做很容易引起衝突，例如當同時有兩個程式同要在同一個位址上填值時，它們不當會改寫對方的資料，。但是程式若以合法的方式要求

在某一個地址作存取時，作業系統應該要儘量滿足這種要求，解決的方法就是讓程式看到的位址不再是實體記憶體的位址，程式可以要求存取任一個可能的記憶位址 (稱爲邏輯位址)，但是當它把這個要求送出來時，作業系統和硬體的配合，會把這個位址轉換成作業系統之前爲它保留的記憶區塊裡 (實體位址)，所以對不同的程式，儘管存取同一邏輯記憶位址，作業系統會把它們指到不同的實體記憶位址上，因此不再有衝突。這整過程對程式而言都是透明的，它們不知道位址被改過。這種轉換的機置會讓系統的效率降低，但是卻是多工系統不可或缺的。

分頁是以頁爲記憶體管理的單位，目前大多數 CPU 都支援這種機制，在 x86 平台上，一頁是 4K bytes，在 alpha 平台上是 8K bytes，因此在 x86 平台上， 64MB 的記憶體相當於 16000 頁。分頁機制是讓記憶體的配置以頁爲單位，例如程式向作業系統要 5K bytes 的記憶空間時，作業系統就會給它兩頁的記憶體 (因爲一頁太少了，不夠用)。很明顯地，這種機制非常浪費 (在上面的例子中，有 3K bytes 的記憶體浪費了)，因此 Linux 2.2版後提供另一種稱爲 SLAB 的機制，它爲許多小量的記憶體或常用的資料結構分別成立一些 cache，因此當程式要求這些記憶體時，作業系統就給它合適的記憶體，不必再給一整頁的記憶區塊，因而提高記憶體的使用效率。分頁機制亦是常見實作虛擬記憶體的方式，它讓邏輯位址和實體位址的轉換以頁爲單位，當程式向作業系統要求記憶體，作業系統在給予某頁時，同時也把邏輯位址和實體位址的轉換建立起來。

爲再加提高效率，程式向作業系統要求記憶時，作業系統會先向程式回應 OK，但並不馬上配置記憶體，直到程式眞的要存取這塊記憶體了，作業系統才會配置這塊記憶體。

現在我們可以回應本節開頭提到的記憶體管理的主要任務：

1. 記憶區塊是以頁爲單位提供給程式使用。

2. 記憶體的保護也是以頁爲單位，例如頁甲是屬於程式甲，程式乙無權存取。

3. 記憶體的共享亦是以頁爲單位。

Linux 的記憶體管理中還有一個很重要的機制，稱爲記憶置換 (swapper)，這是當實體記憶體不足時，例如原本記憶就不多或是正在執行一個大程式，如果又有程式要求配置記憶體，作業系統會把不常存取的記憶頁的內容先存到硬碟裡，把這頁實體記憶體拿來滿足程式的要求。之後若有程式要存取被置換到硬碟的那一頁記憶體內容，作業系統就會把它讀回來，但是放在那裡？是否須要放到原來的實體位址上，不用，那裡有空位就放那裡，只要把邏輯位址和實體位址的對應改一下即可，這裡可以看出虛擬記憶體的好處，若是實體記憶體仍然吃緊，找不出空位，那只好找另一個不常用的記憶頁，請它暫時到硬碟去，位置空出來給這筆急用的資料了。

檔案系統

Linux 支援非常多檔案系統格式，包含 DOS 的 FAT 等數十種不同的檔案系統，但是對應用程式而言，看到卻是統一的介面，不論它讀寫的是 DOS 的 FAT 或是 ext2fs，讀寫都是呼叫相同的函式，它甚至不知道正在讀寫的是那一種檔案系統。

這主要歸功於 Linux 的檔案系統分成上下兩個部份，上面的稱爲虛擬檔案系統 (virtual file system)，它提供給應用程式一個統一的介面，這個介面是個樹狀結構，從根目錄 /開始，所有的檔案都包含在它層層的子目錄下，這些檔案不只包含一般存放在硬碟裡的正常檔案，還有一

些特別的檔案，例如在 /dev/ 下的檔案不是正常檔案，而是用來和驅動程式溝通的虛擬檔案，你在 /dev/mouse 這個檔案中寫入 1，這個 1 不會進到硬碟裡，而是傳給 mouse 的驅動程式。在目錄 /proc/ 下的也全是虛擬檔案，目的是用來和作業系統核心溝通。把驅動程式模擬成一般的正常檔案是 Unix 的特色，事實上讀寫 modem 和讀寫檔案在較高層次上來說也很類似，而且這種統一的作法使作業系統看起來更一致，容易管理。

如上所述，虛擬檔案系統包含一般放在硬碟或光碟之類儲存媒體裡的正常檔案，網路上別部機器提供的網路檔案系統，如 NFS，代表驅動程式的特殊檔案 /dev/*，及其它一些特殊用途的檔案，如 /proc/*。這些全放在根目錄下的目錄結構裡，由虛擬檔案系統控制，大家可以猜的出來，虛擬檔案系統的核心只是個切換器 (switch)，當我們要讀某個檔案時，例如讀 /dev/mouse，虛擬檔案系統會先解析路徑 /dev/mouse，當它找到 mouse 這個項目時，它知道它是周邊裝置的特殊檔案，就把這個讀的命令丟給相對應的驅動程式。

Linux 檔案系統的上層是虛擬檔案系統，下層便是個別檔案系統的實作，每個檔案系統都提供給虛擬檔案系統一個固定的資料結構，資料結構包括這個檔案系統的參數，如一個 sector 有幾個 bytes，另外還包含一組標準呼叫函式，如 read 和 write。當虛擬檔案系統收到一個 read 命令時，它先找出對應的檔案系統，再呼叫這個檔案系統提供的 read 函式，把相關的參數傳過去。

檔案系統可以隨時掛上或卸載，前提是核心支援這個系統，當作業系統啓動時會先掛上根檔案系統 (root)，通常的 ext2fs 格式，之後會再掛上 /etc/fstab 中指定的其系統。

週邊裝置管理

請參考下一節 Device Driver

8.1.4 相關網站及參考資料

1. Linux 核心原始碼[163]。
2. Linux 核心進展新知[164]。
3. Linux 核心程式師新手網站[165]。這個網站由許多 Linux kernel maintainers 共同 support，主旨在幫助培養 Linux 核心程式師，內含許多寶藏，如數百頁的 kernel API 手冊， Alan Cox 寫的 example mouse driver 等等。
4. Linux kernel internal[166].
5. Understanding the Linux kernel, Daniel BOVET & MARCO CESATI, O'REILLY 詳細而且整理得不錯，主要講解 implementation，不提 kernel programming， 但缺網路系統， based on Linux 2.2.x
6. Linux device drivers, ALESSANDRO RUBINI, O'REILLY 這本書是 Linux driver 的經典，第二版已經以 GPL 全文公開在網路上[167] 第一版 based on Linux 2.0.x，第二版 based on linux 2.4.x
7. Kernel projects for Linux, GARY NUTT, Addison Wesley 這本書先簡介 Linux kernel，然後由簡入深，提供一些習題來熟悉 kernel programming。本書對 kernel 本身的講解有限，作習題時最好同時參考 Understanding the Linux kernel。 based on linux 2.2.x
8. Linux IP Stacks commentary, STEPHEN T. SATCHELL & H.B.J. CLIFFORD, CORIOLIS 很多講 Linux kernel 的書都不太提網路系

統，這本書爲網路系統的重要函數作解讀，一方面瞭解網路系統，另一方面可以用來練習讀原始碼。不過它主要只寫到 IP層，但只要同時讀 Linux device driver 的第十四章 network driver，相信不用多久便可以把第二層和第三層串起來。我提這點是因爲第二層和第三層這邊有很多有趣的事可以試，如 netfilter, firewall, QoS, ..., based on Linux 2.0.x 有點舊。

9. 討論核心發展的 mail list: linux-kernel@vger.rutgers.org

10. The Linux Kernel book, Remy Card & Eric Dumas & Frank Mevel, WILEY 作者亦是 ext2fs檔案系統的作者，法國人，特色是先介紹 system call，再慢　　慢一步步深入相關核心實作的細節，因此可以把核心的運作和 system call合起來， 比較具體，缺網路系統。based on Linux 2.2.x

8.1.5 Linux 的延伸與發展

在上面已經提到過， Linux 早已被移植到大多數的桌上型和伺服器平台，所支援的平台都列在 /usr/src/linux/arch 中， Linux 2.4版加入了 IBM S/390， Intel IA64 及 SuperH 的支援。但是除了這些平台外，Linux 現在被延伸應用到更廣的方面或更新的平台，例如嵌入式系統，或即時作業系統等等，由於族繁不及備載，這裡列出相關網站，有興趣的人可以自行查詢：

1. 特定平台資訊

   - ARM[168].
   - uCLinux[169].
   - Open Cores[170].

- IA64[171].
- UltraSPARC[172].

2. 特殊應用

- Linux Router Project[173].
- FirePlug Linux for firewalls and routers[174].
- KOSIX for Linux-based Kiosks[175].
- Embedded Debian[176].

3. 即時作業系統：

- RTLinux[177].
- FSMLabs[178].
- RTAI derivatives/variant of RTLinux[179].
- EL/IX new Cygnus RT API proposal[180].
- SRT-Linux "soft real time" Linux add-on[181].

8.1.6 BSDi

BSDi 包括 FreeBSD[182, 183]， NetBSD 和 OpenBSD，他們都是以 4.4BSD 爲基礎發展起來的，其中以 FreeBSD 的使用者最多，是僅次於 Linux 的開放式作業系統。 FreeBSD 比 Linux 更強　調可靠性，目前主要的應用範圍較集中於網路伺服器。

FreeBSD 起源於 Bill Jolitz 等研究人員發起的 386 BSD 計劃，目的是將 4.3BSD Net/2移植到 80386平台上，到了 1993 年，更多的研究人員加入這個計劃，但 Bill Jolitz 突然決定退出，但是三名研發者 Nate Williams, Rod Grimes 和 Jordan K. Hubbard 決定繼續這個計劃，並改

名爲 FreeBSD，於 1993 年發表 1.0 版，但因爲與 AT&T 的 Unix 版權糾紛， FreeBSD 停頓了好一陣子，直到 1995 年才發表第二版，這是一個全新的 4.4BSD Lite。之後 FreeBSD 加快腳步研發，目前發表了 FreeBSD 4.3 版。

FreeBSD 也是分爲開發版和穩定版，開發版如 FreeBSD4.0-current，穩定版 FreeBSD4.0-stable。

參考資料：

1. FreeBSD 入門與應用，作者　李建達，博碩文化出版。

2. FreeBSD 入門與應用，王波　編著，百資從大陸引入。

3. The implementation of the 4.4 BSD Unix Operating System, Kirk McKusick & Marshall Kierk..., Addison-Wesley.

4. [184] BSD新聞的發佈站台。

8.2 驅動程式

作者：李春和

8.2.1 簡介

何爲驅動程式？ 簡而言之，驅動程式乃是一組程式碼負責將應用程式的需 求，如讀寫等動作，正確無誤地傳給相關的硬體，並使之作出正確的反應。 驅動程式像是一個黑盒子，它們隱藏了硬體的工作細節，應用程

式只需透過一組標準 化的介面，便可心想事成，換句話說，將這組介面對應 (mapping) 到實體裝置 便是驅動程式的工作內容。

Linux上的驅動程式

Linux 作業系統支援上百種驅動程式，幾乎會每一種實體 裝置，如網路卡，繪圖卡，皆需有一個對應的驅動程式，不但如此，功能相同製 造商不同的裝置也需有不同的驅動程式，再者，即使功能、製造商皆相同的裝置， 在不同的作業系統時，如 Linux 與 FreeBSD，亦需有另一組驅動程式。在自由 軟體大行其道， Linux 與 FreeBSD 熱門之際，各家硬體製造商無不希望自己的 產品也能與這些作業系統相容，以佔有市場利基，況且在這個求變的時代硬體淘 汰率何其快，因此，寫驅動程式的專家幾乎不怕沒事幹。

模組化

Linux上大部分的驅動程式皆以模組化 (module) 的形式呈現，但一些必用的裝置驅動程式，如鍵盤、硬碟，則是一開始便載入核心成爲核心的一部分。模組化是 Unix 的重要特色，此架構允許系統在運作期間 (runtime) 擴充核心程式碼，給予使用者極大的彈性，也可降低核心的大小，是一個很好的架構。當使用者需要新的驅動程式時，不須重新啓動系統，只須載入正確的模組，便可隨心所欲地操作硬體，相當方便。

8.2.2 實體裝置的類別

在 Linux 中，一般把裝置分成三個型態，即字元式裝置、 區塊式裝置和網路介面，以下便根據這三個型態詳細說明。

字元式裝置

1. 字元式裝置驅動程式 字元式裝置可對其存取如同一般檔案，而這些對字元式裝置的存取功能便市由字元式裝置驅動程式實作出來的，通常這類驅動程式會提供開、關、讀、寫這四個標準系統呼叫。字元式裝置的特性是其傳輸資料的方式是一次只傳一個位元組的資料,而且是次序性 (sequential)的而非隨機的 (random)，字元式裝置本身通常只是資料傳輸通道，此類的裝置如終端機 ,鍵盤 ,滑鼠等。

2. 主碼與次碼 撰寫驅動程式的第一步是要先定義好這個驅動程式要提供給使用者介面怎麼樣的機制或功能，字元式裝置的存取是透過檔案系統中名字，這些名字有別於一般的檔案名稱，他們被稱爲特殊檔案或裝置檔案，他們也是檔案系統的一個節點 (node)，傳統上是放在 /dev 這個子目錄下，若在這個子目錄下鍵入 'ls -l' 的指令，字元式裝置可由第一行的 'c' 字母來辨認，同樣的，在這個指令下，可看到由逗號分隔的兩個數字，這兩個數字分別代表著個別裝置的主碼 (major number)和次碼 (minor number)，主碼主要用來辨別某一裝置所對應的驅動程式，而次碼是代表相同主碼下 (用相同驅動程式)的不同裝置，核心在開啓時會用到主碼，然後分配執行給合適的驅動程式，事實上，核心根本用不到次碼，它只單純地將次碼傳給相對應的驅動程式，驅動程式收到核心傳來的次碼後，便可以據此知道該驅動那一個裝置，同一驅動程式控制多個裝置是常有的事，次碼可讓驅動程式區別各個裝置。

3. 引入驅動程式 在 2.4 版的核心引入一新的特色 –裝置檔案系統 – devfs，這個檔案系統會使得裝置檔案的管理變得更容易 ,不過 ,目前這個特性還不是預設值。若不用 devfs，要增加新的驅動程式到系統中，驅動程式必須指定一個主碼，並要求作業系統同意，這些動作

需在驅動程式進行初始化時完成，即呼叫定義在 <linux/fs.h> 的函式：

int register_chrdev(unsigned int major, const char *name, struct file_operation *fops);

其中的 major 參數是被要求的主碼， name 參數是裝置的名稱，而 fops 參數是函式指標陣列的指標，其本身是一個跳表 (jump table)，這是用來喚起裝置驅動程式的進入點，如前述，次碼不用傳給 register_chrdev。 一旦裝置驅動程式被暫存到核心表 (kernel table)，它的所有操作便和對應的主碼結合，當需要操作一個字元式裝置時，核心會依據檔案內的主碼，然後在跳表 (jump table)裡找到對應的 fops，借此呼叫正確的驅動程式。 接下來是如何給應用程式一個可以對驅動程式提出要求的名稱，這個名稱需插入在 /dev 下，並給予一組主碼和次碼，要在檔案系統中產生裝置節點，可用 mknod 命令， 但須有 superuser 的權限，例子如下：

mknod /dev/tty c 4 64

4. 移除驅動程式 若欲自系統中移除字元式驅動程式，可輸入以下的指令：

   int unregister_chrdev(unsigned int major, const char *name)；

   移除字元式驅動程式後，主碼也跟著被釋放。核心會根據 major 參數找到先前註冊的裝置名稱，並與 name 參數作比較，若不一樣則傳回 -EINVAL； major 參數超過主碼許可範圍，一樣則傳回 -EINVAL。當自系統中移除字元式驅動程式，卻未能將主碼一併註銷時，系統會發生嚴重錯誤，此時即使用 '救援 '模組，也無法將錯誤復原，因此當謹慎 major 參數和 name 參數的使用。

5. 裝置代碼 每當核心呼叫驅動程式時，主碼和次碼合成的單一資料

型態，足讓驅動程式知道該處理那部裝置，此合成的資料型態放在 inode 資料結構下的 i_rdev ，經由 inode → i_rdev 便可取得此合成的裝置代碼。傳統上， UNIX 會把裝置代碼宣告成 dev_t，而 Linux 則將其宣告成 kdev_t。驅動程式並不理會裝置名稱，它在乎的是裝置代碼。

6. 檔案操作 一開啓的字元式裝置在內部是由 file 結構識別，而核心則使用 file_operations 結構存取驅動程式內的函式，這個結構定義在 <linux/fs.h> 內，它是一個函式指標陣列，每一個檔案都對應到一組函式，這些操作主要是負責實現系統呼叫，因此常被稱爲 open、 read 等，事實上，也可將這些檔案視爲物件，而在其上操作的函式視爲方法，可看到 Linux 也有物件導向的觀念。傳統上，file_operations 結構被稱爲 fop，這是 register_chrdev 呼叫的一個參數。

   字元式驅動程式有其特殊的讀寫函式，此部分待高手完成。

區塊式裝置

如同磁碟一般，區塊式裝置本身就可以當成某種檔案系統，區塊式裝置指的是那些裝置其資料存取是以固定大小的區塊，且是隨機性的，典型的此類裝置便是硬碟和光碟機，區塊的大小取決於不同的磁碟，一般是 1024B，不過，它一定是 2 的指數倍，相較於字元式裝置介面的乾淨和易於使用，區塊式裝置介面，不幸的，是有點亂，核心的發展者總喜歡抱怨這一點，首先，區塊式裝置介面一直都在每一個 Linux 版本的重要位置，而且已證實很難改變，再者，是它的效能，緩慢的字元式裝置已難令人接受，緩慢的區塊式裝置更是拖慢整個系統的元兇，因此區塊式裝置介面的設計往往需考慮到速度的要求，不單只是功能。

1. 主碼與次碼 字元式裝置的開、關、讀、寫等操作，一樣可以應用到區塊式裝置，只是其存取資料大小和方式不同。如字元式裝置，區塊式裝置驅動程式也是由一組主碼辨別，當然，區塊式裝置的主碼和字元式裝置的主碼是完全分開的，比如，區塊式裝置可和字元式裝置同時擁有相同的主碼 32，作業系統卻不會誤認，因爲它們的範圍是分開的，相同的，在 /dev 下鍵入 'ls -l' 可在第一行看到代表區塊式裝置的 'b' 字母，也可查到每一個區塊式裝置的主碼和次碼 (如果有多個裝置共用一個驅動程式的話)。

2. 註冊或移除驅動程式 要註冊或移除區塊式裝置驅動程式的指令如下 : int register_blkdev(unsigned int major, const char *name, struct file_operation *fops)； int unregister_blkdev(unsigned int major， const char *name)； 這兩個函式內之參數意義和用法，與字元式裝置完全相同， 區塊式裝置驅動程式無需專屬的讀寫函式，作業系統提供一套通用的 block_read和 block_write， Linux將區塊式裝置的資料視同爲檔案階層架構 (file hierarchy)，所以才會提供一套通用的讀寫函式，這樣就可借由緩衝處理 (buffering)，達到較佳的效能，而字元式裝置的資料屬於字元式裝置驅動程式本身，因此無一套統一的作法，也無法借由緩衝提昇效能。

網路介面

在討論過字元式裝置和區塊式裝置後，讓我們進入迷人的網路世界，事實上，網路介面是 Linux 裝置的第三種標準類型，讓我們看看它如何與核心的其它部分互動。不同於字元式裝置和區塊式裝置，網路裝置並不依附在檔案系統中，而是在核心層就直接處理封包的傳遞與接收，因此也不爲行程所開啓的檔案所限制。

1. 網路介面和掛載的區塊式裝置的異同 網路介面的角色和已掛載的 (mounted)區塊式裝置相似，區塊式裝置暫存它的一些特色在 blk_dev 陣列和其它核心結構內，然後它便可以根據需求傳送與接收區塊資料；網路介面也以類似的方式運作，它會暫存資料在特定的資料結構，當需要跟外界交換封包時便可正常運作。 雖說類似，已掛載的磁碟和封包傳遞介面仍存有一些不同之處，首先，磁碟是以特殊檔案的形式存在於 /dev 的目錄下，而網路介面則無如此的進入點，由於網路介面不是資料流導向的裝置，因此無法輕易地依附在檔案系統的節點上，檔案系統的一般操作，如開、關、讀、寫，當應用至網路介面時並不合理，網路介面和應用程式之間的資料傳遞不是透過和檔案相關的標準系統呼叫，而是像 socket()、 bind()、 listen()、 accept()和 connect()這類的系統呼叫，所以在 Unix系統裡"任何東西都是檔案"的觀念，在網路裝置並不適用；再者，區塊裝置驅動程式只對來自核心的要求 (request)有反應，然而，網路驅動程式必須非同步 (asynchronously)地接收外界的封包，因此，區塊裝置驅動程式會"被要求"送一個緩衝到核心；相對地，網路驅動程式會"要求"將進來的封包推向核心，應付網路的核心介面其操作模式的設計是不同的。

2. 網路介面的名稱與功能 通常網路介面是指硬體裝置，且但也有力例外，如 loopback 介面便只是軟體介面；每個網路介面都有一個專屬的獨特名稱，如 eth0、 eth1 和 ppp0 等，核心中的網路子系統負責驅動網路介面，而網路介面則實際負責封包的傳遞。網路驅動程式也必須支援一堆的系統管理工作，如設定位址、修改傳輸參數和流量統計，網路使用者介面 (API)自然會反應這等需求。 Linux 核心的網路子系統是按著與協定無關的理念設計的，這適用於網路協定層 (如 IP和 IPX)和硬體協定層 (如 Ethernet和 Token ring)，網路

驅動程式和核心間的互動可以適當地一次處理一個網路封包，這使得協定的議題和網路驅動程式切的乾乾淨淨，協定也不用管實體層傳輸。

3. 註冊網路驅動程式 網路驅動程式會探測 (probe)出網路裝置及硬體位置，如 I/O 埠和 IRQ 線；網路驅動程式是透過模組初使化 (module initialization)的函式 init_module 來註冊，此方法與字元式裝置及區塊式裝置驅動程式有明顯不同。網路驅動程式不需要主碼及次碼，而是爲每一個偵測到的介面，插入一個資料結構到網路裝置的全域列表中，且每一個介面都是由 struct net_device 的資料項來描述。 偵測網路裝置應該在初使化 init 函式時執行， init 收到的單一參數是一個裝置被初使化的指標，傳回值 0 代表成功，錯誤碼爲 -ENODEV。

4. 傳遞與接收封包 網路驅動程式最重要的工作便是資料的傳輸與接收，每當核心需要傳遞封包時，它會呼叫 hard_start_transmit 函式，令其將資料擺在出境佇列 (outgoing queue)中，而每個核心掌控的封包都會被放在邏輯插槽 (socket)緩衝區結構內 –struct sk_buff，其定義可在 <linux/skbuff.h>找到，邏輯插槽 (socket)時常代表網路連結，在網路層每個網路封包都屬於一個邏輯插槽，而任一個邏輯插槽 (socket)的 I/O 緩衝區都列在 struct_sk_buff 中。 接收資料時，得先分配 sk_buff，且在中斷處置常式 (interrupt handler)內換手給網路上層，接收資料一般是透過中斷的方式，除非介面是純軟體的，像前述的 loopback 介面 ,當然寫一個輪詢 (polling)方式的驅動程式也可行 ,但中斷式的驅動程式在效能和計算需求上仍勝過輪詢式。

5. 中斷常式 通常網路介面會對處理器送出中斷訊號，有兩個可能的事件，一是新封包到達，另一方面則爲送出的封包已完成，平常的中斷常式 (routine)只要檢查實體裝置上的狀態暫存器，即可分辨出

是中斷是出於新封包到達或是送出的封包已完成。

參考資料：[185, 186]

1. Linux Driver, author: Alessandro Rubini, published by O'Reilly

2. Understanding the Linux Kernel, author: Daniel P.Bovet &Marco Cesati, published

第九章

中文方案

9.1 國際化與本土化

作者：謝東翰

當我們在發展程式時可能會遇到如下的問題：當一個程式已按照該地區慣用的編碼系統發展完成，已經可以順利處理該地區的文字語言了。如果這時國外，或使用其他語文的地區發覺這個程式很好用，而要將它改成適用於他們地區的語文時，一個很常見的做法就是設法取得該程式的原始碼，然後逐行去改，將隱藏在原始碼中的文字處理、訊息顯示等部分全部改成當地的語文與編碼系統，最後還要重新編譯程式，才能讓它上路。

在這裡很明顯我們見到一個問題，要讓一個程式可以適用於當地的語文與編碼系統，必須直接去改程式碼。萬一該程式正持續發展中，未來出新版本時，整個修改工作勢必又要從頭來過一遍，相當費時費力，同時也很不容易追趕程式原作者的發展腳步。

爲了解決此問題，在 UNIX 的世界中，便逐步形成了「程式國際化」與「資料本土化」的標準，讓世界各地的程式開發者得以遵循。在

此標準之下，程式碼只要寫過一遍，就可以適用於所有的語文與編碼系統，只要系統有支援該語文與編碼系統所需的「本土資料」即可。簡而言之，這裡所採取的概念就是「程式」與「資料」分離並分開維護的方式。

「程式國際化」簡稱 I18N，其意爲 InternationalizatioN 一字中頭尾字母 "I" 與 "N" 中間夾 18 個英文字母，故名。它是在系統底層的函式庫 (即 libc 函式庫) 中實作一組標準的函式介面，可以讓程式存取該地區語系的種種資訊。有了這些資訊，程式本身不僅可以不需要修改，就足以處理各國的語文，同時程式本身甚至連各地區語文的各項細節 (如編碼方式 .... 等) 都不需要知道，因爲這些全部都是由系統函式庫提供的。

「資料本土化」簡稱 L10N，其意爲 LocalizatioN 一字中頭尾字母 "L" 與 "N" 中間夾 10 個英文字母，故名。它是將地區語文的各項細節資料分門別類，安裝在系統底層的資料庫中，以便讓系統函式庫存取，以提供給上頭的應用程式使用。這些用來描述各地區語文的資料，我們稱之爲「區域化資料庫 (locale)」，它們包括以下的類別 (categories):

1. LC_COLLATE: 該地區文字排序規則，以及正規化表示式 (regular expression) 的比對依據。
2. LC_CTYPE: 該地區所使用的編碼系統、字集、與文字分類、轉換等資訊。
3. LC_MESSAGES: 各應用程式區域化的訊息顯示。
4. LC_MONETARY: 該地區所通行的貨幣格式。
5. LC_NUMERIC: 該地區所通行的數字表示格式。
6. LC_TIME: 該地區所通行的時間、日期表示格式。

對於同一個地區語文而言，除了 LC_MESSAGES 之外，其他所有的類別都是固定的，不會隨應用程式的不同而有改變。故這些類別的資料就只需準備一份即可，它們是由系統底層函式庫直接提供，可以讓所有的應用程式分享。至於 LC_MESSAGES 訊息顯示的部分，由於各應用程式的訊息都不同，故這部分的資料是由應用程式提供。

在這些類別中，決定一個程式是否在該地區已「本土」化的最重要因素，一是 LC_CTYPE，二是 LC_MESSAGES。前者賦與程式處理該地區文字的能力，後者賦與程式用該地區的語文來顯示的能力，故底下我們就針對這兩個類別作更進一步介紹。

1. LC_CTYPE 類別：

   LC_CTYPE 與文字處理有關，它讓程式可以在不知道該地區所採用的編碼細節之下，進行文字的處理。它包括以下的資訊：

   a) 該地區所使用的字集。所謂「字集」就是指一群文字符號的集合，換言之就是該地區會用到的所有文字符號。

   b) 該字集所採用的編碼方式。一般而言，一個編碼方式所能容納的字數是固定的，故當我們談到某個編碼系統時，會自然而然將它所代表的字集聯想在一起。但稍後我們將看到，其實二者的觀念是分開的，特別是在 GNU/Linux 的系統中 (使用 glibc-2.2.X)。

   c) 該字集中每個字的分類與轉換。這一點其實在使用拼音文字的語系中比較有意義，而在使用象形文字的地區 (如亞洲語系) 則意義比較不那麼大。這裡指的是拼音文字中字母大小寫的轉換、是不是數字 (阿拉伯數字或十六進位數字)、是不是標點符號、是否可以印出、是否爲電腦控制碼 .... 等等。由於電腦的發展起源於使用拼音文字的西方世界，故當他們在制定 C 語言

等標準函式庫介面時就特別將這一點列入考慮。顯然這些分類轉換規則會隨著地區語系的不同而不同。而在使用象形文字的地區，也許沒有所謂大小寫的分別，但仍然可以大至依既定的規則填入分類表中。

d) 多位元組字元與寬字元的轉換。這是特別爲亞洲地區，使用大量非拼音文字的地區而制定的。因爲這些地區所使用的文字符號數目，大大超越了單一一個位元組 (即 8 位元) 所能表示的範圍，因而往往需要依一定的編碼規則，將多個位元組合起來代表一個字。然而，在程式中往往不會有這些編碼規則的資訊，這使得該程式無法從一個位元組串列中辨認出一個一個的文字。因此系統就提供了這一組多位元組字元與寬字元的轉換機制。由於每個寬字元所佔的位元組數是固定的，因而一個寬字元就代表一個文字，不管它原先在多位元組字元形式時是由幾個位元組所組成的。因此，程式本身可以完全不需要知道各地區的編碼方式，就能採這種方式逐字處理文字，而這些詳盡的編碼辨識與轉換工作就全部交由系統底層函式庫來完成了。這正是程式國際化最重要的關鍵。

2. LC_MESSAGES 類別 :

此類別與程式訊息顯示有關。由於各程式會顯示的訊息都不相同，故這些訊息必須由程式本身提供，而不能像其他類別一樣由系統函式庫提供。在一個國際化的程式中，程式訊息的部分是與程式碼本身分開維護。在實作上，我們將程式所有的訊息集中放在一個文件檔中，而該文件檔的訊息開始時只會用程式原作者慣用的語言來表示。如果我們希望該程式也能顯示其他語文的訊息時，我們只需要去做翻譯的工作即可，而不必眞的去修改程式碼本身。因此，訊息翻譯與程式維護可以分頭進行，翻譯的工作不需要由程式原作者、

或有經驗的程式設計師來做，只需他熟悉該語文，並對該程式有一定的熟悉度即可。故基本上，任何人都可以參與翻譯的工作。

當程式編譯安裝完成後，已翻譯成各國語文的訊息檔也會一併安裝入系統的區域化資料庫中。當程式啓動，需要做訊息顯示時，它會呼叫系統提供的函式介面，依目前的語系設定 (見後述) 來正確抓取該語文的訊息並顯示出來。萬一目前的語系設定找不到相對應的訊息翻譯檔時，則程式會自動以其原始的語系來顯示。

9.1.1 區域化資料庫名稱和語系設定

各地區所屬的區域化資料庫名稱格式如下：

{語系名}_{地區名}[.{編碼系統名}]

其中 [.{編碼系統名}] 有時候會省略。以我們台灣地區所使用的爲例：

zh_TW.Big5

其意即爲「中文語系」 (zh)「台灣地區」 (TW)「使用 Big5 編碼系統」。如果將後頭的 [.{編碼系統名}] 省略掉，就是這個樣子

zh_TW

那這樣的區域化資料庫是用什麼樣的編碼系統呢？就依不同的 UNIX 系統而有不同了。在大部分早期的 UNIX 系統中，用的可能是 EUC-TW 編碼系統，即 ISO11643，而在 GNU/Linux 系統中，用的則是 Big5，原因是 Big5 是目前台灣地區使用最廣泛的編碼。

如果我們不特別做語系的設定，則程式在啓動時，會以系統預設的語系來運作，一般而言其區域化資料庫名就是 "C" 或 "POSIX"，也就是原始 C 語言所採用的編碼系統 (ASCII) 與英文訊息等等。如果我們希望改變程式運作的語系，則我們必須在程式啓動前先做好環境語系的設定，也就是設好各類別的環境變數。例如：

```
LC_CTYPE=zh_TW.Big5; export LC_CTYPE
```

```
LC_MESSAGES=zh_TW.Big5; export LC_MESSAGES
```

經過如上的設定後，則之後我們在此 shell 下啓動程式時，程式就可以處理台灣地區的 Big5 文字，同時訊息顯示也會是台灣地區 Big5 中文。萬一我們希望程式可以處理 EUC-TW 的文字，但仍以 Big5 中文顯示訊息時，就這樣設定：

```
LC_CTYPE=zh_TW.euctw;      export LC_CTYPE
LC_MESSAGES=zh_TW.Big5;    export LC_MESSAGES
```

而其他類別的設定方式依此類推。

在很多情況下，我們通常會希望一口氣將所有的類別設定成相同的語系，也就是讓我們的整體環境全部處於同一個語系下。當然我們可以用上述的方式一個個類別分別設定，但除此之外系統還提供了另外兩個環境變數，以方便我們的作業。一是 LANG，另一個是 LC_ALL。例如我們這樣設：

```
LC_ALL=zh_TW.Big5;     export LC_ALL
```

其效果就完全等價於將所有的類別全部設定了。而 LANG 的用法也是一樣，所達到的效果也類似，但意義稍有不同，這裡要留意優先順序的差別。一般系統對這些環境變數的優先順序是：

```
LC_ALL  >  LC_*  >  LANG
```

也就是說，任何一個 LC_ 類的變數設定後，會使 LANG 的設定的相對應類別失效。如果我們完全不設任何的 LC_ 類的環境變數，只單單這麼設

```
LANG=zh_TW.Big5;     export LANG
```

則所有的類別都會以 LANG 的設定來運作，除非我們特別去設了某個 LC_ 的環境變數，如此這個類別就會以新的設定來運作 (但其他的類別不變)。相似的道理，如果我們設了 LC_ALL 的環境變數，則所有的類別設定，包括 LANG 的設定全部會失效，而改以 LC_ALL 的設定來運作。

9.1.2 GNU/Linux 下的實作方式

長久以來，中文化的問題一直是兩岸三地 GNU/Linux 使用者所面臨的一大挑戰。在過去我們所謂「中文化」的方式，就是將需要處理中文的程式，如終端機、文字編輯器、郵件程式等的原始碼拿來，一行行改，改成可以處理我們的 Big5 碼。然而之前我們也提過，這樣的方式相當費時費力，而且每個程式都要修改，也難以跟上國外程式發展團隊的腳步。

直到近兩三年來， GNU/Linux 的 I18N 與 L10N 環境發展開始起步，我們也才能依循這些標準，逐步建構我們的中文環境。我們認爲，遵循這樣的標準來建構中文環境，才是徹底的解決之道，我們不能自己關起門來，自行實作一套獨特中文環境，因爲這不僅事倍功半，同時國外的程式開發團體也無法直接支援我們。只要按照 I18N 的標準來發展程式，不僅可以馬上使用我們的中文，而且也可以在別的語系下使用。

GNU/Linux 的 I18N 環境的發展，到了去年中 glibc-2.2 系列正式問世後，才算完全成熟。它不僅完全支援 Unicode 環境，同時在 I18N 與 L10N 方面還擁有許多先進的特色，茲簡述如後：

1. glibc 內部的編碼轉換系統 (iconv) 擁有共同的「基底字集」，該基底字集採用 UCS4 編碼，目前仍持續擴編當中，理論上將可以函蓋世界上所有已知的編碼系統的轉換對應。透過基底字集，可以達到完善的轉碼機制。同時，由於在各語系下其多位元組編碼轉成寬字元時，其實就是轉成 UCS4，這在將來 Unicode 通行時，將有利於資料的處理。

2. glibc 內部擁有數量龐大的編碼轉換表，大部分是各編碼系統與 UCS4 的轉換用，也有一些是用於不同編碼間直接轉換。所有的轉換表皆採動態模組載入的方式供應用程式使用，故只有在需要時系

統才會自動載入所需的轉換表來執行，使用完畢後可以卸下，不會浪費記憶體空間。

3. glibc 擁有完整的 I18N 程式呼叫介面，以及完整的 Unicode 輸出入與處理的呼叫介面。

4. 在區域化資料庫方面， glibc 將「字集」與「編碼」的概念分開。一個區域化資料庫有一個明確的字集，代表這個地區語文可能會使用到的所有文字符號，而這個字集可以自由選擇一個編碼系統來套用。例如我們台灣地區的字集包含所有的中文字 (簡繁體都有)，如果我們選用 Big5 編碼，則區域化資料庫名稱就是 `zh_TW.Big5`；若我們選用 EUC-TW 編碼，則名稱就是 `zh_TW.euctw`；當然我們甚至可以選用 GB2312 或 Unicode 等編碼來套用。

   然而，要注意的是，由於各編碼所能包含的字數是固定的，同時它們也只能包含特定的字集，因此，僅管我們的中文字集包含了所有的中文字，但一旦選定編碼系統後，其所能使用的中文字自然僅限於該編碼系統所包含的範圍。因此，假如我們選用了 Big5，就表示在此環境下我們無法使用簡體字；反之若選用 GB2312 亦然。

5. glibc 中各區域化資料庫預設的編碼系統與傳統的 UNIX 系統稍有不同。所謂的「預設編碼系統」指的是在 "{語文名}_{地區名}" 這樣的區域化資料庫名稱中，所採用的編碼系統。傳統的 UNIX 系統中所採用的預設編碼系統多半是依照官方或業界的標準，例如在台灣地區就是 EUC-TW。而在 glibc 中，則是以當地最廣泛流通的編碼系統做爲預設，故在台灣地區就是 Big5。

   更進一步地， glibc 會自動將未登錄的編碼系統名稱，改以其區域化資料庫的預設編碼系統來取代。例如我們將語系環境設定爲 `zh_TW.euctw`，由於 "euctw" 在 glibc 內部已有登錄，故它就會採用

EUC-TW 做爲此環境下的編碼系統。萬一我們將語系環境設定爲 zh_TW.unknown，由於"unknown" 一字沒有登錄，故 glibc 會自動以 zh_TW 預設的編碼系統 Big5 來取代。

6. 在訊息顯示方面，各應用程式的訊息翻譯只需依各語文地區分別保留一份即可，不需要分別爲不同的編碼系統都保留一份。以台灣地區中文爲例，我們只需要一份 zh_TW 的訊息即可，至於它是用 Big5 寫成的，或 EUC-TW 寫成的都沒關係。假如它原來是以 Big5 寫成的，但我們卻希望以 EUC-TW 來顯示時， glibc 會自動爲我們做好轉碼的工作。

   也許有人會問，那我們只需要保留一份 zh (即中文) 的訊息就好了嘛，這樣豈不是兩岸三地都可使用了？然而這麼做並不恰當，原因是兩岸三地因歷史背景與文化的隔閡，使得它們各自發展成獨特的語文，或稱「地區性方言」，對同一個句子的翻譯，可能兩岸三地的翻法都各自不同。故我們不能單純用轉碼的方式，直接將台灣地區的翻譯轉成簡體字供對岸使用，而必須爲他們分別保留一份他們自己的訊息翻譯。

7. 在訊息翻譯的維護工作，以及系統函式的呼叫介面上，各 UNIX 系統的實作方式都不盡相同。而在 glibc (或所有的 GNU 系統、程式) 中，則統一使用 GNU gettext，以方便程式撰寫以及後續的翻譯維護。

8. 在區域化資料庫中，除了上述那幾個傳統的類別之外，新一代的 glibc-2.2 還擴增了以下的類別 :

   ```
   LC_PAPER, LC_NAME, LC_ADDRESS, LC_TELEPHONE,
   LC_MEASUREMENT, LC_IDENTIFICATION
   ```

   這些是根據 ISO/IEC JTC1/SC22/WG20 N690 (1999/06) 的新規範

而來，用以描述更多的地區性慣例，如住址格式、電話號碼格式、抬頭稱位 .... 等等。

9.1.3 X Window 的國際化環境 (Xi18n)

Libc 的國際化與本土化是整個 UNIX 系統的語文處理的基礎，其他應用程式都是以此基礎出發往上延伸發展，圖形介面的 X Window 系統亦是如此。同樣的， X Window 系統所面臨的課題也是在於文字處理與訊息顯示這兩大方面。就訊息顯示方面而言，過去在 libc 的 I18N 與 L10N 標準尚未確立前，曾一度使用資源檔 (X Resources) 來存放訊息的翻譯，唯此方式普及率不高。現在有了 I18N 與 L10N，只要使用 LC_MESSAGES 類別，就能達到所需的目標。

但在文字處理方面就複雜許多，主要是文字的輸出與輸入兩方面。在圖形介面中，不同的文字需要不同的字形來顯示，故爲了要完整顯示一個地區所有的文字符號，往往需要同時使用許多不同的字型才能達成。故在 X Window 的區域化資料庫定義中就多了一個字型集 (FontSet) 的定義。以我們的 zh_TW.Big5 區域化資料庫爲例，在使用 XFree86 的 X Window 系統中 (包括 GNU/Linux)，我們的字型集就包括了一個 ISO8859-1 的字型，以及一個 BIG5-0 的字型，分別用於顯示英文與 Big5 中文。但在其他使用非 XFree86 的 UNIX 系統中，其字型集的定義則各有不同。

有了字型集的定義， X Window 系統就會依據 libc 的 LC_CTYPE 編碼辨識與分析機制，自動從字型集中挑選適當的字型來顯示各個文字。因此， X Window 的應用程式同樣也不需要知道各語文的編碼細節，使用字型集就能自動完整地顯示出該語文中所有的文字符號。因而程式可以國際化。

至於文字的輸入方面比起文字顯示還要更複雜些，這裡牽涉到各語系的輸入法問題，故在此 X Window 系統特別定義了一個 XIM (X Input Method) 的協定來做處理，它同樣是以 libc 的 LC_CTYPE 類別爲基礎運作的。有關這方面的細節，請參閱下一節「中文輸入」的說明。

參考資料：

1. man setlocale, man locale

2. info libc

3. ISO/IEC JTC1/SC22/WG20 N690: [187]

4. Unicode: [188]

5. Xlib Programming Manual, 作者：Adrian Nye, 出版者：O'Reilly, ISBN 1-56592-002-3

9.2 中文輸入法

作者：謝東翰

由於一般電腦鍵盤上的字鍵數目有限，多半 (遠) 少於地區語文所需的文字或符號的數目，使得在很多情況下無法做到一個按鍵就對應一個「字」或符號。對於使用拼音語系的地區而言，由於基本的拼音字母較少，因此可以藉由功能鍵改變鍵盤狀態，讓同一個字鍵在不同狀態下輸出不同的字母。例如在我們一般的鍵盤中，CapsLock 關閉的情況下可以打出小寫英文字母，反之可以打出大寫英文字母。

然而，到了非拼音語系的地區，特別是亞洲語系的國家，光是藉由上述方式也不足已打出所有的文字與符號了，故在這些地區中通常需要藉助「輸入法程式」來完成文字輸入的工作。一個輸入法程式主要的工

作是攔截使用者所敲入的字鍵序列，經由查表或其他方式將這些字鍵序列轉換成該地區文字符號，然後才輸出至應用程式。而字鍵序列與文字符號之間的對應關係，就形成了各式各樣的輸入法。而逐一將字鍵序列組合出一個個文字符號的過程，就稱之爲「組字」。

大體而言，我們的「中文輸入」就是以上述的方式達成的。以下我們就針對 GNU/Linux 系統來談中文輸入的解決方案。

1. 純文字模式下的中文輸入：

   在早期電腦的記憶體與運算能力仍十分有限的時候，若要執行 X Window 系統多半無 法達到令人滿意的效能，故那時候的操作環境多半是在純文字模式之下，而純文字模 式下的中文輸入方案也應需求而一一出現。

   在此模式下，由於一般的文字終端機只能顯示英文，故剛我們嘗試用中文輸入法程式來輸入中文時，我們必須連帶要設法讓文字終端機也能顯示中文，如此才有意義。因此，在此模式下的輸入法程式通常會與一個可以顯示中文的文字終端機連在一起，啓動它後就等於進入了一個「中文輸入輸出」的環境。而且，一個能顯示中文字的中文終端機程式，往往比輸入法本身還要重要。

   一般而言，文字模式下的輸入法並沒有特殊的規範或協定，程式所要做的，只有取得使用者的字鍵輸入，再將中文輸出到「標準輸出 (standard out)」管道，系統自然會將這些文字餵入應用程式中。只要應用程式能夠接受並處理 8 位元字元碼，則不會有任何問題。

   這類文字模式下的中文輸入法與中文終端機程式種類很多，早期以王佑中先生所發展的 chdrv 爲主要的代表，它是一個 GNU/Linux 下的中文終端機、輸入法的程式。另一個在同時期的類似程式爲 yact, 它與 chdrv 的不同點在於 chdrv 是直接去控制 tty 裝置以模

擬出中文終端機，而 yact 卻是以顯示卡繪圖函式庫 svgalib 爲基礎，將純文字模式的終端機轉爲圖形模式以顯示中文字。這些早期的中文輸入系統各自有其缺點，而隨著時間的流逝也漸漸沒有在維護了，但其他類似的中文軟體卻相繼出現。它們或是以早期的中文輸入系統爲基礎，針對其缺點加以修正，或是引進新的設計概念，使其有較好的功能與較多的輸入法支援。這類的軟體包括 bcs16、 jmce、 jmcce .... 等。至於 FreeBSD 系統上，有早期常見的 big5con，以及後來出現的可顯示、輸入 GB 碼的中文終端機程式 cce .... 等。

這類用於純文字模式下的中文輸入系統，由於需要將原本的文字顯示轉爲圖形顯示，如此才能畫中文字。然而轉換成圖形顯示的方式可能依系統的不同而有不同，例如同樣在 x86 系統下執行的 GNU/Linux 與 FreeBSD 系統，它們在處理終端機程式的方式不盡相同，而且對硬體顯示卡的控制方式也有差異，這使得分別針對這些系統所設計的中文終端機程式不容易移植到其他系統上去，甚至同樣在 GNU/Linux 平台上，爲 x86 系統所設計的也無法在其他硬體平台上編譯後執行。這一點與以下將介紹的 X Window 環境下的中文輸入法程式有很大的不同。儘管如此，這些不同的中文輸入系統仍在原始碼的層次上互相截長補短，彼此吸收對方的優點。例如最近才出現的 jmcce, 它就是改進自 bcs16、 cce、與 jmce 等中文輸入系統，而這也正是自由軟體世界中分享、互惠精神的一個很好的範例。

2. X Window 下的中文輸入 :

   a) XIM 協定 :

   由於近來電腦硬體的進步，使得執行 X Window 系統漸漸可以達到令人滿意的效能，同時在一般情況下使用圖形操作介面往

往比純文字的操作介面方便，故有越來越多的使用者轉向使用 X Window 的環境，使用 X Window 下的中文終端機程式與中文輸入法。因此在相對上，純文字模式的中文輸入法使用者較少了，而 X Window 下的輸入法發展則日漸蓬勃起來。

在 X Window 的圖形介面底下，由於各程式間的交互作用是以「視窗」爲單位，它們可以藉由標準的 X 協定達到彼此間的溝通，故我們自然而然地可以將中文終端機程式與輸入法本身分開發展，而不再需要像純文字模式下將二者綁在一起 (然而在早期的確有將二者綁在一起的解決方案，如 cxterm，它是直接自 X Window 的標準終端機程式 xterm 修改而來)，如此在一個圖形桌面上只需執行一個中文輸入法，就可以對許多中文終端機程式提供中文輸入的服務，不但節省系統資源，同時模組化的開發模式也讓後續維護工作容易得多。

然而，在 X Window 下的輸入法所面臨的問題卻比純文字下要複雜許多。由於考慮到程式國際化等方面的問題，故 X Window 定義了一組標準的輸入法協定，稱之爲 XIM (X Input Method) 協定。此協定是架構在程式國際化 (I18N) 與系統區域化資料庫 (locale) 之上的，故只要遵守此協定，則應用程式就可以在不需修改程式碼的原則下，接受來自各種語系輸入法程式的文字輸入。

與純文字輸入方式不同的是，在這裡輸入法程式不預先攔截使用者的字鍵輸入。而應用程式與輸入法程式之間的關係，就好像客户端與伺服端一樣，應用程式提出輸入請求，則輸入法程式提供輸入服務。因此，當我們對一個視窗做中文輸入時，實際上敲入的字鍵是直接送往應用程式本身，而應用程式在處理它之前，會先經由 XIM 協定將這些字鍵序列送往輸入法程式，然後由輸入法程式那邊取得中文字。故在此協定下，應用程式

又稱之爲 “XIM client”，而輸入法程式又稱之爲 “XIM server”。

在一個 “X Window 的顯示設備 (display)” 中 (這裡意指一個螢幕、一個鍵盤，再加一個滑鼠，也就是一個 X Window 終端桌上環境)，可以同時執行好幾個 XIM server，它們可以輸出不同語系的文字 (例如有的可以用來打中文，有的可以打日文)，或其中有幾個可以輸出相同語系的文字。而 XIM client 要選那一個 XIM server 來使用，必須透過以下兩個環境變數的設定 :

```
export LC_CTYPE={區域化資料庫名稱}
export XMODIFIERS="@im={XIM server 名稱}"
```

其中前者指定了語系環境，後者指定了要用那一個輸入法程式。所有的 XIM client 在啓動之前必須先有上述的環境變數設定，啓動後才能接受該 XIM server 的輸入。

一般 XIM server 所顯示的資訊可以分成以下三類 :

i. 組字資訊 : 顯示於 XIM server 組字的過程中。

ii. 狀態資訊 : 顯示 XIM server 目前的狀態。

iii. 其他輔助資訊 : 例如功能表選單或線上文件說明等。

其中依組字與狀態資訊顯示的位置不同，就形成了各種操作介面 (input style)，供使用者方便使用。其中包括 :

i. Root: 此二資訊都顯示在 XIM server 的主視窗內。

ii. OverTheSpot: XIM server 會在 XIM client 的輸入游標附近開啓一個小視窗，以顯示組字資訊。如此使用者在打字過程中眼睛就不需要時時去看 XIM server 主視窗的組字資訊。

iii. OffTheSpot: XIM client 會在自己的視窗中開出一塊區域，讓 XIM server 來顯示其組字資訊。此區域通常是在 XIM client 視窗的底下。

iv. OnTheSpot: XIM server 提供必要的資料給 XIM client，讓

它用自己的方法來畫 XIM server 的組字資訊。這通常是給有特別需求的 XIM client 選用。

其中 Root 模式是最簡單的方式，一般而言所有的 XIM server 與 XIM client 都會支援，至於其他的模式則不一定。必須 XIM server 與 XIM client 都同時支援的模式才能使用。如果二者同時支援幾種模式，則當二者開始連繫，準備讓使用者輸入時，它們就會先協調好，以挑選最佳的模式來用。當然，很多時候使用者也可以在 XIM client 這邊指定要使用那一種模式。

由於 X Window 環境下有較好的人機介面，因此大部分的輸入法程式都是在 X Window 下運作的。早期的輸入法程式可能不支援 XIM 協定，但近年來新發展的輸入法程式幾乎都支援 XIM 協定，以下就幾種常見、可以做中文輸入的輸入法程式稍作介紹。

b) XCIN (X Chinese INput method):

XCIN (http://xcin.linux.org.tw) 可能是台灣地區最普及的一個輸入法程式。它最早期是由 劉德華 (Edward Der-Hua Liu) 先生發展的 (1994 年 10 月)，那時它並不支援 XIM 協定，直到 1999 年全新改寫後才開始支援。歷經多年的延革，並加入了多位網友的貢獻，使它逐漸成爲一個穩定且功能齊備的輸入法程式。其特色包括 :

i. 支援 BIG5, BIG5HKSCS, 與 GB2312 等多種編碼方式。使用時只要在不同的區域化資料庫環境 (LC_CTYPE) 下啓動它，即可自動採用該語系的編碼來做輸入。

ii. 支援動態外掛式輸入模組，讓我們可以依需要開發不同的輸入法模組，以支援不同的輸入法。目前 XCIN 套件內含三個輸入法模組 :

A. zh_hex: 內碼輸入模組。

B. gen_inp: 一般性輸入法模組，以查表方式進行組字，適用於字根類型的輸入法。

C. bimsphone: 自動猜字注音輸入法。它以 libtabe ([151, ?]) 做爲猜字引擎，經由字詞的分析辭庫查詢完成自動猜字動作。

除此之外，由其他計畫所發展的 XCIN 輸入法模組計有：

A. 酷音中音輸入法 (http://chewing.oio.cx/) 它是由中研際資訊所徐讚昇博士指導、龔律全與陳康本所設計的輸入法模組。 它也是一個自動猜字的注音輸入法，擁有比 bimsphone 更方便的功能，使用 上與 Microsoft 平台上常見的自然輸入法很接近。

iii. 支援多種輸入法。除了上述的內碼輸入與自動猜字注音輸入法以外，還支援了：

A. 倉頡輸入法 (BIG5, BIG5HKSCS, GB2312)

B. 行列 30 輸入法 (BIG5)

C. 拼音輸入法 (BIG5, GB2312)

D. 簡易輸入法 (BIG5)

E. 一般注音輸入法 (BIG5, GB2312)

F. 雙拼輸入法 (GB2312)

G. 五筆輸入法 (GB2312)

另外，一些商業公司也提供了適當的輸入法表格，在其特定的版權限制下，可以在 XCIN 底下使用。其中包括了大易輸入法、嘸蝦米輸入法 .... 等等。

iv. 支援 Xi18n, XIM 協定與 Root 和 OverTheSpot 輸入模式。

v. 擁有豐富多樣使用者自定選項。

vi. 可以跨平台編譯執行，其中包括 GNU/Linux, FreeBSD 與 HP-UX。

當然， XCIN 仍不是一個完美的程式，未來仍有相當大的改善空間，主要包括 : Unicode 與更多編碼系統的支援、輸入法模組功能改進、可同時處理多區域化資料庫的 XIM client 請求、更多 UNIX 平台的移植等等。

c) Chinput

這是一個針對 GB 編碼與大陸地區使用者習慣而設計的中文輸入法，其設計者爲于明儉先生。它是以 cxterm 爲基礎發展出來的。它同樣支援 XIM 協定，與 XCIN 相較，它具備較佳的拼音輸入功能，有較好的圖形操作介面，同時也支援 GB、 Big5、 JIS (一種日文的編碼) 與 KS (一種韓文的編碼) 等編碼方式，以及許多常見的輸入法表格和方便的輸入功能，是一支相當優秀的中文輸入法程式。

參考資料 :

1. Xlib Programming Manual [189]
2. XCIN 發展計畫 [190].
3. libtabe 計畫 [151, 191].
4. 酷音輸入法計畫 [192].
5. Chinput [193].
6. 中文 HOTWO [194, 195].

9.3 中文輸出

作者：謝東翰

9.3.1 簡介

本小節的重點，主要在中文文件的排版與列印方面。在 UNIX 世界中，長久以來已 不乏排版與列印的工具，例如功能強大的幕後排版系統 TeX/LaTeX，用以預覽、顯 示、以及列印的文件圖形化語言 Postscript，以及在自由軟體世界中常見的 Postscript 解釋器 Ghostscript 及其附帶的印表機與其他週邊裝置的驅動程式 .... 等等。然而，要讓這些排版與列印工具可以處理中文 (或其他多位元組的文字) 卻 是一項難題，主要是在於文字的編碼方式以及字型方面。

就以中文爲例，由於在一般的電腦裝置 (如印表機) 或電腦軟體並不內建數量龐大 而且筆畫複雜的中文字型，故當我們要做中文輸出時，必須要有特別的處理方式。 例如我們希望在印一份純文字的中文文件，我不能直接將它送交純文字印表機去列 印，因爲這些印表機多半認不得中文編碼，也沒有字型可用，故它最多只能將送來 的中文編碼拆成一個個連續的 ASCII 或 ISO8859-1 的編碼，再用其內建的 ASCII 或 ISO8859-1 字型印出來，結果就是印出一連串的亂碼。

爲了解決此問題，通常當我們要做列印時，不論是印已編排好的文件或純文字，我 們多半要做如下的處理 : 辨認出文件內的所有文字，中文字的部分就去找適當的字 型，將代表該字的字型 (或說該字的「圖案」) 畫在該文件內，最後處理完後原本 的文件 (或文字檔) 就成爲一個「圖形檔」，最後才將該「圖形檔」送印。因此， 在很多情況下，印一篇中文的文字檔案，其實是印一份內嵌了中文字型的圖形檔案。

圖形檔的格式有許多種，但在自由軟體的世界中，其最終輸出的格式通常是 Postscript。 故以下我們就以此爲中心，簡介自由軟體世界的中文輸出方案。

9.3.2 圖形輸出格式：Postscript 與 PDF

Postscript 是早年 Adobe 公司與頻果電腦公司所合作開發的一種描述式語言。該 檔案的內容看起來就是一大堆奇奇怪怪的語法與關鍵字 (也就是說，它看起來像一個 純文字檔)，描述了一張紙上那個座標要畫一個點、那裡要畫一條線、線的粗細、顏 色、還有那裡要擺一個文字 (這裡指多半的是 ISO8859-1 的文字)、字的大小、形狀、 字型 .... 等等。而這些描述式的語言，必須要由懂得此語言的程式或週邊設備來讀， 才能從中組合出一幅完整的圖案。之所以用 Postscript 做爲最終的送印圖檔格式， 我認爲主要是因爲它有很好的可移植性 (因爲它在一般人或程式看來只是純文字檔， 而不是二進位檔)，而且其語法有很高的彈性與延展性，使得我們可以輕易將圖案縮放 而不至於失眞。同時， Postscript 也是業界文件輸出格式的重要標準，早期有許多學 術論文與排版印刷工作的最終輸出都是 Postscript，甚至有許多印表機直接就可以讀 懂 Postscript, 故它們可以直接列印 Postscript 的文件，而不需要額外的驅動程式。

另一個讓自由軟體世界採用 Postscript 爲最終輸出形式的重要理由，是因爲 Ghostscript 軟體。它是一個可以讀懂 Postscript 語言的解譯器，更重要的是它內 含許多驅動程式，可以將 Postscript 文件轉換成各種形式輸出，包括許多印表機的 驅動程式，使得這些不懂 Postscript 語言的印表機得以順利列印文件。除此之外， Ghostscript 還有 X Window 的繪圖模組，可以讓 ghostview、 gv 等程式在 X Window 的環境下顯示 Postscript 的文件；它還有可以將 Postscript 轉換成其他圖檔格式 (jpeg, png, tiff, ...)、甚至可以送交傳眞的 faxg3 格式 .... 等等 (因此，當 我們要用 modem 傳眞文件時，我們可以先將該文件的 Postscript 檔案準備好，經 由 Ghostscript 程式轉換後送出)。故 Ghostscript 可以說是一個功能相當強大的 Postscript 轉換引擎。

由於 Postscript 只是一種圖檔格式而已，除非我們熟悉它的語法，否則我們很難直接編寫、修改 Postscript 的文件，也無法輕易在 Postscript 文件中搜尋文字，更難以用滑鼠剪貼的方式將 Postscript 文件中的文字剪貼到別的視窗中。爲了彌補這些缺點，故近年來 Adobe 公司又另外發展了一套稱爲 PDF 的文件。它同樣包含了 Postscript 所有的優點 (然而它是個二進位檔，而非純文字檔，這一點與 Postscript 格式不同)，也彌補了 Postscript 的不足，其中包括 :

1. 由於 PDF 的文件有壓縮過之故，使得它的大小比起相同的 Postscript 文件要小得多。

2. 在 PDF 流覽器 (如 xpdf) 中顯示 PDF 文件時，速度比起用 Postscript 流覽器 (如 gv) 來顯示 Postscript 要加快不少。

3. 用 PDF 編輯器可以很容易地直接修改 PDF 文件，也可以直接將 PDF 文件中的 文字直接剪貼到別的視窗中。

4. PDF 文件還支援索引、 hyper-reference、及 bookmark (目錄) 的功能。

由於這些優點，故在這一兩年來它的普級率迅速上升，許多原本採用 Postscript 的學 術文件也逐漸改採 PDF 了，連近年來新版的 Ghostscript 軟體也能讀懂 PDF 的格式。 然而在自由軟體世界中，現階段 PDF 仍然不是最終輸出的格式，特別是在印表方面， 很多時候仍然要先轉換成 Postscript之後才能送印，而許多繪圖程式的列印輸出也多半還是 Postscript 而非 PDF，我想主要是因爲對自由軟體世界而言它還太新，還需要 一段時間才會有完整的支援。

9.3.3 中文字型的轉換與內嵌

不論是 Postscript 文件或 PDF 文件，都可以直接內嵌所需的字型，也可以只提示所 需的字型名稱與每個字在字型檔中的編碼索引等。後者的檔案大小當然比前者來得小， 但前提是文件內所使用的字型名與字型規格要有一個通用的標準，如此該份文件才能在 各種環境下閱讀與列印。故一般只包含 ASCII 或 ISO8859-1 文字的 Postscript 文件 是不內嵌字型的，但是一份中文文件在目前常見的情況下是必須內嵌字型的，其原因正 是目前還沒有通用的中文字型與規格。

在早期 (也許 PDF 格式尚未問世的時候)，用做內嵌的中文字型來源是一種名爲 HBF 的點陣字型，當時有許多程式可以將 HBF 字型轉換成 Postscript 可以使用的格式， 然而這類的點陣字型解析度有限，故內嵌在 Postscript 文件中往往不夠美觀，特別是 當我們要做有限度的文字縮放時更是慘不忍睹。

後來，由於 TrueType 字型逐漸普及，同時可以處理 TrueType 字型的引擎 —- freetype 函式庫的問世，許多基於該函式庫的 TrueType 字型轉換程式一一開發出來， 使得它成爲 Postscript 與 PDF 內嵌字型的最重要來源。內嵌了 TrueType 轉換字型 的文件不但比起過去要美觀許多，且由於 TrueType 本身就具有可縮放的特性，故我們 可以輕易產生高解析度的轉換字型，使得文件在有限度的縮放下仍不失眞。同時由於文 鼎公司對自由軟體世界的支持，慷慨地捐贈了兩套 TrueType 中文字型供大家自由使用， 讓我們在中文文件輸出的問題上得以徹底解決。

然而，最近我們開始慢慢體認到，使用內嵌字型的文件輸出仍然不是最佳的解決方案 (然而，對許多人或許多應用場合而言，它已經是相當不錯的解決方案了)。最佳的解決 方案仍然是希望以不需內嵌中文字型的方式，其優點除了可以讓輸出的文件檔變小以外， 同時還可

以保證在任何尺度上的縮放而不失眞。由於內嵌在文件中的字型在很多情況下 是將 TrueType 字型轉成點陣字型之後，才進行內嵌 (例如在 CJK-LaTeX 所編譯的文件 中內嵌了由 TrueType 轉成 PK 的點陣字)。僅管由轉換過來的字型有相當高的解析度， 但再高的解析度也是有限的，故在做大尺度的縮放時仍然會失眞。還有一點就是，就目 前的測試結果，若在 PDF 文件中使用內嵌字型則無法自 PDF 文件中剪貼文字到別的視 窗或文件中，其原因可能不單純，也有可能是 PDF 文件中的字型編碼沒有符合 Adobe 的規格所至。

要做到眞正不內嵌中文字型於文件中，有許多工作要做，其中包括 :

1. 要有一套統一的字型名稱與規格 : 這可能是最令人頭疼的部分，因爲這裡卡到了授 權問題。由於 Postscript 與 PDF 文件格式是 Adobe 公司開發的，故其所用的字 型名稱都會灌上 Adobe 之名，而這些名稱是否能在自由軟體世界中，以自由軟體的 遊戲規則來運行，將會是個問題，其中尤以 PDF 最爲嚴重。同時，在定這些名稱與 規格時，我們不能關起門來自己定，必須同時與許多團體談，可能包括 Adobe 公司、 TeX/LaTeX 開發團隊、 Ghostscript 開發團隊、甚至其他與排版、字型相關的計畫與 商業公司等。

2. 要有一個字型的來源 : 這一點在現階段已有初步成果，目前我們已有將 TrueType 字型轉換成 Postscript 所需的 Type1 字型的工具程式，例如 ttf2pt1 或 chpfb, ttf2pfb 等。除此之外，另外還有一套稱爲 t1lib 的函式庫，可以用來進一步處理 Type1 字型，例如將字縮放、旋轉，或將它轉換成點陣的格式以用於其他的用途 .... 等等。

3. 如果我們有了 Type1 的字型，我們還可以進一步地將它合成 Type0 的 CID-Keyed 字型，可用於 Postscript 與 PDF 的文件中。這一點目前仍在討論階段，但相當具 有可行性。理論上，只要透過字型名

的 alias (別名) 機制，我們也許就可以用此 技術直接顯示 Adobe 做出的不內嵌中文的 pdf 檔了。而如何定訂此 alias 的標準， 仍需進一步討論。

4. 我們的文件轉換引擎：Ghostscript 要能處理多位元組編碼、並利用這些字型：這一 點目前也正積極開發中，例如下一代的 Ghostscript 就有處理多位元組編碼的能力， 除了迎接未來的中文 Type1 字型以外，它還內含了一個可直接使用 TrueType 字型 的模組，可以做到不需要字型的內嵌，就可以使用 TrueType 字型來顯示文件，其原 理是將 TrueType 字型模擬成 Type0 CID-Keyed 字型來用。但這可能會面臨可攜性 的問題，萬一將此文件拿到其他沒有安裝此 TrueType 字型的系統下就無法讀了。因 此，它目前也同時在開發直接內嵌 TrueType 字型的技術，做爲因應此問題的配套措 施。

9.3.4 中文 Postscript 與 PDF 文件的產生

在 UNIX 或自由軟體的世界中，有許多與文書處理與繪圖有關的程式可以產生 Postscript 的文件，因爲當文件要送印時就需要 Postscript。至於 PDF, 在自由軟體世界中則比較 少，主要見於文書排版方面。以下就以中文文件的輸出爲中心，茲簡介如下：

純文字的轉換

目前較通用的純文字檔轉 Postscript 格式的程式爲由 python 語言所寫成的 bg5ps 程式，它所產生的 Postscript 文件採內嵌中文字型的方式，而中文字型的來源即爲 TrueType 字型。其實， bg5ps 還有許多其他功能，包括簡單的格式化與排版的指令、 同時可以在未內嵌中文字型的 Postscript 中加入內嵌字型。其中後者在早期的中文 列印尤爲有用。

早期有些應用程式如 Netscape，它可以在視窗中顯示中文，讓我們觀看中文文件。但當它要將文件輸出成 Postscript 格式時，因爲沒有標準的字型可用，而它也不會去產生內嵌字型，故最後的輸出結果就是將每個中文字拆成一個個連續的 ASCII 或 ISO8859-1 的字元，也就是一堆亂碼。這時，bg5ps 程式就可以扮演一個過濾器的角色，它在這些亂碼的 Postscript 文件中安插入內嵌字型，於是一份可正常顯示中文的文件就產生了。可做到類似 bg5ps 的功能的程式還有許多，在此就不一一詳列。

至於將純文字檔轉成 PDF 格式，有一個很類似 bg5ps 的程式 —- bg5pdf，它一樣也是以 python 語言寫成的 (外加 PDFLib 函式庫)，然而與 bg5ps 不同的是，它在文件中只標示了中文字型的名稱 (爲不內嵌的 Acrobat Reader 的中文 CID Keyed 字型)，故由它產生的文件可以享有 PDF 文件所有的好處。而在現階段，它所產生的文件可以用 xpdf 程式來流覽，只要經過適當的設定，xpdf 會自 X Window 系統中抓取適當的字型來做顯示，故可以避開沒有 CID Keyed 字型的問題。但長遠來看，我們仍然需要一套完整的 CID Keyed 字型，如此才能完整解決中文 PDF 文件的顯示問題。

CJK-LaTeX

TeX/LaTeX 是一套功能相當強大的幕後排版系統，它使用類似程式語言的方式，來格式化並編排文件，並配合 Postscript 的輸出，可以輕易達到高品質文件的輸出要求。特別是它容易處理數學符號的特性，使得它特別適於學術文件、書籍、出版的排版工作。

要讓 LaTeX 能處理中文，必須在 LaTeX 系統上額外加裝一套 CJK-LaTeX 的巨集。顧名思議，此巨集可以付與 LaTeX 處理中文、日文、韓文等編碼系統的能力，而它也漸漸成爲 LaTeX 的標準之一。而它最

後在產生 Postscript 文件時，是採用內嵌 中文字型的方式，其字型來源也是 TrueType。它利用 freetype 函式庫的衍生工具 ttf2tfm、 ttf2pk 將 TrueType 字型轉換成 TFM 與 PK 點陣字型，前者用於前段的 排版工作，後者則用於最後的字型內嵌。

以 LaTeX 編排的文件也可以轉換成 PDF 格式，有兩種方式：一爲 pdftex, 它是 TeX/LaTeX 程式的變形，專門用來輸出 PDF 格式。另一種方式爲使用 dvipdfm 程 式，它可以將 TeX/LaTeX 編排過程的中間產物 —- DVI 檔轉換成 PDF 格式。所 謂 DVI 意指 DeVice Independent，它是 TeX/LaTeX 編排之後的輸出，可以用來轉 換成各種所需的格式，包括顯示的螢幕上預覽 (使用 xdvi 程式)、轉換成 Postscript 格式 (使用 dvips 程式)、或丟給特定印表機的驅動程式送印 .... 等等。

不論是 pdftex 還是 dvipdfm, 現階段所產生的 PDF 文件都是內嵌字型的文件。其 內嵌的字型可以是 PK 字型、或 TrueType 字型、或者是 Type1 字型 (若內嵌了 PK 字型則會有大尺度縮放失眞的問題，因爲 PK 字型是點陣字型，但若內嵌其他二者則 不會)。其中 dvipdfm 目前內嵌 TrueType 字型還不夠成熟，相信在不久的將來就會 有成果。

LaTeX 的外衣 —- LyX

LyX 是一個以 LaTeX 爲基底的一個圖形化排版系統，讓我們不必須記一大堆 LaTeX 的排版指令，可以直接視覺化地在視窗內編排我們的文件，而最後的排版輸出才交由 LaTeX 處理。它的中文支援目前正慢慢發展中，包括支援 XIM 協定，使得我們可以 用 XIM server 輸入法程式直接在它的視窗內打中文。而它的 CJK-LaTeX 指令的部 分在經過少許的修正後也能正確地搭配使用。

各應用程式與列印系統的整合

如前所述，由於 Ghostscript 可以正確處理 Postscript 與 PDF 的文件，並且可以 將它們轉換成各種格式輸出，包括許多印表機的列印格式。故使用 Ghostscript 正 是最標準且一致的列印解決方案。除了上述可以產生 Postscript/PDF 文件的程式以 外，事實上還有許多與文書工作相關的應用程式，如辦公室軟體 KOffice/AbiWord 等，或繪圖軟體 gimp、xfig、 gnuplot .... 等，它們在做列印輸出時都可以產生 Postscript 的格式 (未來也許會有 PDF 格式)，故只要經過適當的整合，就能順利 解決列印的問題。至於未來當我們慢慢轉形使用「不內嵌」中文字型的文件時，還要 再注意各應用程式所用的字型名稱是否與 Ghostscript 所用的一致，這些都需要更 進一步的協調與整合的工作。

參考文獻 :

1. A First Guide to Postscript [196]

2. Adobe PostScript 3 [197]

3. PDF [198]

4. Ghostscript [199]

5. Other resources [200]

6. TrueType [201]

7. Freetype project [202]

8. 李果正
   CJK-LyX 中使用中文 [203]
   CJK/LaTeX enviroment 中文 Type1 及 TTF 的使用 [204]

9. 字型與文件轉換工具

   bg5ps[205]、bg5pdf[206]、chpfb[207]、ttf2pt1[208]、pdfTeX[209]、dvipdfm[210]。

10. t1lib [211, 212]

11. jmcce [213]

本文內容感謝李果正先生的參與討論與幫忙校對 :-))

附錄 A

Open Public License Draft v0.4

OPEN PUBLICATION LICENSE Draft v0.4, 8 June 1999

I. REQUIREMENTS ON BOTH UNMODIFIED AND MODIFIED VERSIONS

The Open Publication works may be reproduced and distributed in whole or in part, in any medium physical or electronic, provided that the terms of this license are adhered to, and that this license or an incorporation of it by reference (with any options elected by the author(s) and/or publisher) is displayed in the reproduction.

Proper form for an incorporation by reference is as follows:

Copyright (c) 2001 by 中華民國軟體自由協會 (Software Liberty Association of Taiwan). This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v0.4 or later (the latest version is presently available at http://www.opencontent.org/openpub/).

The reference must be immediately followed with any options elected by the author(s) and/or publisher of the document (see section VI).

Commercial redistribution of Open Publication-licensed material is permitted.

Any publication in standard (paper) book form shall require the citation of the original publisher and author. The publisher and author's names shall appear on all outer surfaces of the book. On all outer surfaces of the book the original publisher's name shall be as large as the title of the work and cited as possessive with respect to the title.

II. COPYRIGHT

The copyright to each Open Publication is owned by its author(s) or designee.

III. SCOPE OF LICENSE

The following license terms apply to all Open Publication works, unless otherwise explicitly stated in the document.

Mere aggregation of Open Publication works or a portion of an Open Publication work with other works or programs on the same media shall not cause this license to apply to those other works. The aggregate work shall contain a notice specifying the inclusion of the Open Publication material and appropriate copyright notice.

SEVERABILITY. If any part of this license is found to be unenforceable in any jurisdiction, the remaining portions of the license remain in force.

NO WARRANTY. Open Publication works are licensed and provided "as is" without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement.

IV. REQUIREMENTS ON MODIFIED WORKS

All modified versions of documents covered by this license, including translations, anthologies, compilations and partial documents, must meet the following requirements:

1) The modified version must be labeled as such. 2) The person making the modifications must be identified and the modifications dated. 3) Acknowledgement of the original author and publisher if applicable must be retained according to normal academic citation practices. 4) The location of the original unmodified document must be identified. 5) The original author's (or authors') name(s) may not be used to assert or imply endorsement of the resulting document without the original author's (or authors') permission.

V. GOOD-PRACTICE RECOMMENDATIONS

In addition to the requirements of this license, it is requested from and strongly recommended of redistributors that:

1) If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty

days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.

2) All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.

Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy and CD-ROM expression of an Open Publication-licensed work to its author(s).

VI. LICENSE OPTIONS

The author(s) and/or publisher of an Open Publication-licensed document may elect certain options by appending language to the reference to or copy of the license. These options are considered part of the license instance and must be included with the license (or its incorporation by reference) in derived works.

A. To prohibit distribution of substantively modified versions without the explicit permission of the author(s). "Substantive modification" is defined as a change to the semantic content of the document, and excludes mere changes in format or typographical corrections.

To accomplish this, add the phrase 'Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.' to the license reference or copy.

B. To prohibit any publication of this work or derivative works in whole or in part in standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from the copyright holder.

To accomplish this, add the phrase 'Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.' to the license reference or copy.

OPEN PUBLICATION POLICY APPENDIX:

(This is not considered part of the license.)

Open Publication works are available in source format via the Open Publication home page at http://works.opencontent.org/.

Open Publication authors who want to include their own license on Open Publication works may do so, as long as their terms are not more restrictive than the Open Publication license.

If you have questions about the Open Publication License, please contact TBD, and/or the Open Publication Authors' List at opal@opencontent.org, via email.

附錄 B

作者簡介

- 謝東翰 (Tung-Han Hsieh) thhsieh@linux.org.tw 台大物理系博士班學生， XCIN 計畫 (http://xcin.linux.org.tw) 維護者。研究領域在物理方面主修 Lattice QCD；而在電腦與資訊方面則涉獵了自由軟體的開發與應用、程式國際化 (I18N) 與本土化 (L10N)、中文輸入法、以及平行計算 .... 等。過去亦曾在台大物理系擔任過短期的網管工作。

- 吳鴻煦
  中華電信研究所 , 負責資訊中心。 熟悉網路管理、 Linux IP Layer、網路安全及資料庫應用系統。

- 葉平 (Ping) pyeh@softwareliberty.org, http://www.pingyeh.net/
  台大物理系客座助理教授，軟體自由協會常務理事兼發言人。自 1993 年開始接觸 0.9x 版的 Linux，在美國費米加速器國家實驗室攻讀博士學位時期開始在工作上使用 Linux，並於 1996 年底開始在費米實驗室和中央研究院同時推動以 Linux 做爲大型科學計算平台的 PC Farm 計劃。

- 鄭大慶 (Big) big@icdiy.org
  國立台灣大學電機研究所畢業，十年以上 C/C++ 物件導向語言與應用程式開發經驗，任職於台灣育電科技。目前正以自由軟體爲基礎，進行企業 e 化相關產品設計工作。最新著作：企業入口網站 DIY。

- 李柏鋒 (Informer) pofeng@linux.org.tw
  長庚兒童醫院住院醫師， 1993 年起開始接觸 Linux，業餘自由軟體愛好者。

- 李健秋 (Andrew)

- 陳開基 (Kaiji Chen) kaiji_chen@ezhi.com
  教育部公費留學，法國巴黎第十一大學固態物理博士，目前從事網路研發工作。

- 胡師賢 (Albert) albert@iazone.net
  新廣廉科技執行副總經理，十年的嵌入式系統開發經驗。兩年前開始接觸 Linux，相信以 Linux 爲首的自由軟體對台灣過去以資訊硬體爲主的電子產業，不但有截長補短的產業升級效果，自由軟體的社群發展經驗更是進入知識經濟時代，國內產業人文與觀念提昇的重要參考。

- 李春和 (Daniel Lee) daniel_lee@ezhi.com
  李春和，台大物理博士；目前服務於爭鋒科技，利用 Linux 上所提供的網路功能，進行產品開發，正大享開放軟體的好處。

- 廖志宇 (Armani Liao) armani.liao@ieee.com
  廖志宇，台大夜中文系學生。

參考資料

[1] IBM 的 Lou Gerstner 在 2000 年 12 月 12 日於 eBusiness Conference Expo 的演講，見 http://www.ibm.com/lvg/1212.phtml.

[2] 原文見 Free Software Foundation 的 "The Free Software Definition" 網頁： http://www.fsf.org/philosophy/free-sw.html.

[3] 軟體自由，見 http://www.softwareliberty.org/faq/software-liberty.

[4] Sendmail 網頁：http://www.sendmail.org/.

[5] Berkely Internet Name Domain, http://www.isc.org/products/BIND/.

[6] Gnu Compiler Collection, http://www.fsf.org/software/gcc/gcc.html.

[7] Robert Young and Wendy Goldman Rohm, "Under the Radar," 中譯本：「Linux 紅帽旋風」，天下文化， ISBN: 957-621-665-6[484]。

[8] 微軟公司有關 shared source licensing 的新聞稿，見 http://www.microsoft.com/presspass/features/2001/may01/05-03csm.asp.

[9] Eric S. Raymond, "The Cathedral and The Bazaar," O'Reilly & Associates, Inc. ISBN: 1-56592-724-9. 網頁：http://tuxedo.org/ esr/writings/cathedral-bazaar/.

[10] Sam Palmisan 於 2001 年 1 月 31 日在 LinuxWorld Conference & Expo New York 的 keynote speech, http://www.ibm.com/news/2001/02/conference_trans.phtml.

[11] Linux 中文延伸套件：http://cle.linux.org.tw/.

[12] 開放源碼國際研討會：http://twopensource.org/icos01/.

[13] Z Object Publication Environment：http://www.zope.org/.

[14] 由台北市電腦公會之 Linux 促進會所舉辦 , http://www.tca.org.tw/.

[15] The Jargon file version 4.3.1 http://www.tuxedo.org/~esr/jargon/html/index.html.

[16] Linux Standard Base, http://www.linuxbase.org/.

[17] LSB Common Specification 1.0.0, http://www.linuxbase.org/spec/.

[18] July 4 article of Linux Weekly News, http://www.lwn.net/2001/0704/.

[19] Linux Internationalization Initiative, http://www.li18nux.org/.

[20] CPlant project, http://www.cs.sandia.gov/cplant/.

[21] http://eltoday.com/article.php3?ltsn=2001-05-25-001-14-PS.

[22] http://www.cray.com/news/0101/sc.html.

[23] http://linuxtoday.com/news_story.php3?ltsn=2000-03-25-006-04-PS.

[24] http://www.apache.org/.

[25] http://www.netcraft.com/survey/.

[26] http://www.activemedia-guide.com/messoft_mrkt.htm

[27] http://www.zdnet.com/zdnn/stories/comment/0,5859,2560523,00.html

[28] http://tech.chinatimes.com/newshtm/ppheral/900523110.htm.

[29] http://www.libertytimes.com.tw/today1214/today-e5.htm.

[30] http://www.trend.com .tw/corporate/about/pressscenter/2000news/200004027.htm

[31] http://www.bnext.com.tw/mag/2000_07/2000_07_844.html.

[32] http://www.gnu.org/software/libc/libc.html.

[33] http://www.gnu.org/software/hurd/hurd.html.

[34] http://www.gtk.org.

[35] http://developer.gnome.org/doc/API/liboaf/tutorial.html.

[36] http://www.gnome.org/gnome-office/bonobo.shtml.

[37] http://orbit-resource.sourceforge.net/

[38] http://glade.gnome.org/

[39] http://developer.gnome.org/doc/API/libglade/libglade.html

[40] http://www.gnome-db.org/

[41] http://perso.mandrakesoft.com/~david/toronto/developers/html/slide_3.html

[42] http://developer.kde.org/documentation/books/kde-2.0-development/ch12.html

[43] http://developer.kde.org/documentation/tutorials/xmlui/preface.html

[44] http://www.trolltech.com/products/qt/.

[45] http://developer.kde.org/documentation/library/dcop.html

[46] http://www.xml-rpc.com/

[47] http://developer.kde.org/documentation/kde2arch/xmlrpc.html

[48] David A. Wheeler, "Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!" http://www.dwheeler.com/oss_fs_why.html.

[49] http://www.zdnet.com/sp/stories/issue/0,4537,2387282,00.html

[50] http://www.syscontrol.ch/e/SWePIX/SWePIXe.html

[51] http://www.syscontrol.ch/e/news/Serversoftware.html

[52] http://www.zdnet.com/sp/stories/issue/0,4537,2196115,00.html

[53] http://www.sysadminmag.com/articles/2001/0107/0107a/0107a.htm

[54] http://securityportal.com/cover/coverstory20000117.html

[55] http://www.google.com

[56] http://www.openldap.org.

[57] http://www.enhydra.org.

[58] http://www.jboss.org.

[59] http://www.samba.org.

[60] http://www.tcpdump.org.

[61] http://www.packetfactory.net/libnet.

[62] http://www.packetfactory.net/Projects/Libnids/.

[63] http://www.joedog.org/.

[64] http://sourceforge.net/projects/issl/.

[65] http://www.gnetlibrary.org/.

[66] http://asg.web.cmu.edu/cyrus/.

[67] http://libsocket.sourceforge.net/.

[68] http://www.hylafax.org/.

[69] http://www.sistina.com/gfs/.

[70] http://www.zebra.org.

[71] http://www.gated.org.

[72] http://www.openssl.org.

[73] http://www.openssh.org.

[74] http://netfilter.samba.org .

[75] http://www.interbase.com.

[76] http://www.sap.com/solutions/technology/sapdb.

[77] http://ostenfeld.dk/ jakob/Software-RAID.HOWTO .

[78] http://linuxtoday.com/stories/10698.html.

[79] http://63.219.148.22/nova/man/raw.htm.

[80] http://www.atnf.csiro.au/ rgooch/linux/vfs.txt.

[81] http://web.mit.edu/tytso/www/linux/ext2intro.html.

[82] http://www.linuxgazette.com/issue55/florido.html.

[83] http://www.reiserfs.org/.

[84] http://oss.sgi.com/projects/xfs/.

[85] http://www.gnu.org.

[86] http://sources.redhat.com/binutils/.

[87] http://www.gnu.org/directory/gdb.html.

[88] http://www.gnu.org/software/ddd/ddd.html.

[89] http://www.gnu.org/software/make/make.html.

[90] http://sources.redhat.com/autoconf/.

[91] http://sources.redhat.com/automake/.

[92] http://www.gnu.org/software/libtool/libtool.html.

[93] http://www.gnu.org/prep/standards_toc.html.

[94] http://www.cvshome.org/.

[95] http://www.gnu.org/software/bash/bash.html.

[96] http://www.perl.com.

[97] http://www.perl.org.

[98] http://www.python.org/.

[99] http://www.php.net/.

[100] http://php.resourceindex.com/.

[101] http://www.scriptics.com/.

[102] http://cui.unige.ch/eao/www/TclTk.html.

[103] http://javascript.internet.com/.

[104] http://www.pcwebopaedia.com/TERM/J/JavaScript.html.

[105] http://www.gnu.org/directory/texinfo.html.

[106] http://www-cs-faculty.stanford.edu/ knuth/cweb.html.

[107] http://www.oasis-open.org/cover/xmlLitProg.html .

[108] http://www.gnu.org/software/emacs/emacs.html.

[109] http://www.identicalsoftware.com/xwpe/.

[110] http://apps.kde.com/rf/2/latest.

[111] http://sunsite.dk/GUIDE/.

[112] http://www.borland.com/kylix/.

[113] http://www.codeforge.com/.

[114] http://www.st-andrews.ac.uk/ iam/docs/tutorial.html.

[115] http://www.gnu.org/directory/clibrary.html.

[116] http://www.muppetlabs.com/~breadbox/software/ELF.txt.

[117] http://www.mcsr.olemiss.edu/cgi-bin/man-cgi?elf+3.

[118] http://anubis.dkuug.dk/JTC1/SC22/WG15/.

[119] http://www.linuxguruz.org/foldoc/foldoc.php?XPG.

[120] http://www.gnu.org/software/gcc/libstdc++/.

[121] http://kbs.cs.tu-berlin.de/ jutta/toast.html.

[122] http://www.libsdl.org/.

[123] http://www.lokigames.com/development/smpeg.php3.

[124] http://www.xiph.org/.

[125] http://www.xiph.org/.

[126] http://www.68k.org/ michael/audiofile.

[127] http://www.zip.com.au/ erikd/libsndfile.

[128] http://www.imagemagick.org/.

[129] http://www.libpng.com/.

[130] http://www.libmng.com/.

[131] http://www.libtiff.org/.

[132] http://prtr-13.ucsc.edu/ badger/software/libungif/.

[133] http://www.mesa3d.org/.

[134] http://www.openh323.org/docs/PWLib/.

[135] http://www.vovida.org/protocols/index.html.

[136] http://www.openss7.org/.

[137] http://www.avayalabs.com/project/libsafe/index.html.

[138] http://asg2.web.cmu.edu/sasl/.

[139] http://www.x.org/.

[140] http://www.xfree86.org/.

[141] ftp://ftp.x.org/pub/R6.4/xc/doc/hardcopy/X11/xlib.PS.Z.

[142] http://www.opengroup.org/motif/.

[143] The Definitive Guides to the X Window System Volume 3/3M: X Window User's Guide for X11/OSF Motif Author: Valerie Quercia and Tim O'Reilly. Publisher: O'Reilly

[144] http://www.lesstif.org/.

[145] http://www.kde.org.

[146] http://www.gtk.org/.

[147] http://www.gnome.org/.

[148] http://www.fltk.org/.

[149] http://www.pango.org.

[150] http://fribidi.sourceforge.net/.

[151] http://libtabe.sourceforge.net/.

[152] http://www.nec.co.jp/japanese/product/computer/soft/canna/.

[153] http://www.winehq.com/ .

[154] http://www.sleepycat.com/.

[155] http://freetype.sourceforge.net/.

[156] http://dickey.his.com/ncurses/ncurses.html.

[157] http://xmlsoft.org/.

[158] http://sources.redhat.com/bzip2/.

[159] http://www.s-lang.org/.

[160] http://www.gnu.org/software/pth/pth.html.

[161] http://www.users.dircon.co.uk/ crypto/.

[162] http://www.rasterman.com/.

[163] http://www.kernel.org. Linux 核心原始碼

[164] http://lwn.net. Linux 核心進展新知

[165] http://www.kernelnewbies.org. 核心程式師新手網站這個網站由許多 Linux kernel maintainers 共同 support，主旨在幫助培 養 Linux 核心程式師，內含許多寶藏，如數百頁的 kernel API 手卌， Alan Cox 寫的 example mouse driver 等等。

[166] http://www.moses.uklinux.net/patches/lki.html#toc1.

[167] http://www.oreilly.com/catalog/linuxdrive2/chapter/book/index.html.

[168] http://www.armlinux.org.

[169] http://www.uclinux.org.

[170] http://www.opencores.org.

[171] http://www.linuxia64.org.

[172] http://www.ultralinux.org.

[173] http://www.linuxrouter.org.

[174] http://www.fireplug.net.

[175] http://www.touchdynamics.com/kosix.html.

[176] http://www.emdebian.org.

[177] http://www.rtlinux.com.

[178] http://www.FSMLabs.com.

[179] http://www.rtai.org.

[180] http://Sourceware.cygnus.com/elix/.

[181] http://www.cl.cam.ac.uk/dmi1000/linux-srt/index.html.

[182] http://www.freebsd.org.

[183] http://www.mckusick.com/courses/.

[184] http://www.daemonnews.org.

[185] http://www.freeos.com.

[186] http://www.networkcomputing.com.

[187] http://wwwold.dkuug.dk/JTC1/SC22/WG20/docs/n690.pdf.

[188] http://www.unicode.org/.

[189] 作者：Adrian Nye,
出版者：O'Reilly,
ISBN 1-56592-002-3.

[190] http://xcin.linux.org.tw/.

[191] http://xcin.linux.org.tw/libtabe/.

[192] http://chewing.oio.cx/.

[193] http://www.lslnet.com/linux/ch/chinput.html.

[194] http://www.linuxdoc.org/HOWTO/Chinese-HOWTO.html.

[195] http://www.linux.org.tw/CLDP/Chinese-HOWTO.html.

[196] http://www.cs.indiana.edu/docproject/programming /postscript/postscript.html.

[197] http://www.adobe.com/products/postscript/main.html.

[198] http://www.adobe.com/products/acrobat/adobepdf.html.

[199] http://www.cs.wisc.edu/ ghost/index.htm.

[200] http://www.geocities.com/SiliconValley/5682/postscript.html.

[201] http://www.truetype.demon.co.uk/.

[202] http://www.freetype.org/.

[203] http://www.study-area.org/tips/latex/cjk-lyx.html.

[204] http://www.study-area.org/tips/latex/cjk-ttf.html.

[205] http://students.washington.edu/cschin/bg5ps/.

[206] http://students.washington.edu/cschin/bg5ps/bg5pdf/.

[207] http://download.sourceforge.net/mirrors/turbolinux/pub/turbocontrib-6.0/TurboContrib/SRPMS/.

[208] http://ttf2pt1.sourceforge.net.

[209] ftp://ftp.inet.cz/pub/Mirrors/pdfTeX.

[210] http://odo.kettering.edu/dvipdfm.

[211] ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/software/t1lib/.

[212] http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/rmz/t1lib/t1lib.html.

[213] http://www.t-linux.com.tw/jmcce.