

# 精通 GDB：深入 GNU 開發與除錯工具

Jim Huang(黃敬群/jserv)

<http://jserv.sayya.org/>

email: <[jserv.tw@gmail.com](mailto:jserv.tw@gmail.com)>

<[jserv@openmoko.org](mailto:jserv@openmoko.org)>

文件版次: 0.2 最後更新日期: Aug 17, 2007

## 前言

從事若干項嵌入式系統(Embedded System)與應用程式的軟體開發後，筆者深感 GNU 開發工具<sup>[1]</sup>威力強大且具備高度彈性，不僅在 UNIX(或相容系統)幾乎可到「放諸四海而皆準」的境界，在嵌入式系統開發中，往往 GNU/Linux 的 BSP(Board Support Package)也較其他商業系統，如 QNX、VxWorks，或 WinCE 等，能在前期進行開發與調整，還能銜接豐富的免費與自由軟體資源所提供之技術基礎，儼然成為當今新興的開發模式。

然而，正如 1975 年出版、軟體工程界經典著作的《人月神話》(The Mythical Man-Month)<sup>[2]</sup>在第一章即以史前時代陷入焦油坑的恐龍作為日益失控的軟體開發比喻一般，在今日看來，作者 Frederick Brooks 的說法一點也不為過，貼切地描述強調“Time to Market”或充斥“Me too”思維的專案開發中，面對時程、預算及品質控管的壓力，專案經理人、程式設計師、測試員，以及無數專案關係人持續超時工作、交期延誤、來自上司或客戶沈重的壓力，不啻是陷入一種「焦油坑」嗎？

要掙脫上述的泥淖，除了要有良好的規劃與管理，對工程技術的掌握更是不可或缺，所以即使有成千上萬免費取得、可自由修改的軟體作為基礎，但背後涉及的技術細節就是隱然而生的挑戰，本文即以 GNU 開發工具中極具威力的 GDB(即“GNU Project Debugger”)<sup>[3]</sup>為探討對象，述及如何偵錯、追蹤與分析既有程式，與在嵌入式系統的應用。礙於資源限制，本文僅 GNU/Linux 系統作為主要探討之平台，但歡迎熟稔 FreeBSD、NetBSD、Solaris，或其他開放系統之高手不吝指教與補充。

## 文件標注說明

除了文字與圖表外，本文涵蓋部份程式碼列表與指令使用的互動，在標注上的呈現，前者會予以縮排(indent)，而後者會以**粗體字**標示在終端機或 Debugger session 中，開發者鍵入的指令或操作。在行文中，若有加底線的文字或訊息則表示加強語氣或重點區段，以區分其他內容。

## 參考開發平台與軟體版本

若沒有特別注明，本文之概念性描述應可適用於多數可運行 GNU 開發工具之系統平台，特別是 GNU/Linux，然而為釐清因為軟體版本更動與硬體表現的可能影響，以下列舉筆者之開發平台以茲參考：

- IBM/Lenovo X60 筆記型電腦(處理器: Intel Core™2 1.83GHz)
  - 作業系統: Ubuntu gutsy(開發中版本)
    - Linux kernel Version 2.6.22-9-generic
    - gcc version 4.1.2
    - glibc version 2.6.1
  - GDB Version 6.6-debian
- OpenMoko GTA01 開發平台(處理器: Samsung S3C2410 266MHz)
  - 作業系統: OpenEmbedded OM-2007.2
    - Linux kernel version 2.6.21-moko
    - gcc version 4.1.1
    - glibc version 2.4
- HP 商務桌用型主機(處理器: Pentium-4 3GHz)
  - Microsoft Windows XP

## 為何我該學習 GDB ?

在詳細探討如何應用 GDB 前，筆者試著闡述 GDB 這類 source-level debugger 對於程式開發與維護的重要性。王建興日前於撰文「Google 時代的程式撰寫」<sup>[4]</sup>提到，網路技術與知識累積日益完善的今日，對程式開發而言，可謂邁入新的紀元，無論是開發時程、錯誤排除，以及基於開放原始碼的分享與互助精神，已帶來極大的衝擊，過去即使閉門造車、土法煉鋼數年或數個月，其成果可能抵不上花上一個晚上在網際網路上搜尋相似的開放原始碼專案，略為修飾調整功能這等「偷吃步」的行徑，誠如俗語所說「黑貓白貓，能抓老鼠，就是好貓」，王建興描述這現象對程式設計師帶來的變化：「在過去，對程式員的訓練也許不甚重視追蹤原始碼以及拆解原始碼。大家重視的是如何撰寫可讀性高的程式，以及設計出具彈性、重用性高的程式。但在開放原始碼的世界中，有太多的程式不僅可讀性低，而且不具彈性、重用性也不高—往往只是為了專門的需要而 hack。如果具備了追蹤程式碼的能力，除了可以透過現成的程式碼學習新的技術之外，也可以做為拆解程式碼的基礎設施。而要拆解程式碼的第一步，自然是看懂程式碼，看懂之外，要進一步判斷那些是自己要的，那些不是，同時評估怎麼調整這些程式碼的架構，可以去除掉自己不要的部份，保留自己要的部份，又毋需對現有程式碼做過大幅度的更動」

基於這樣的思維，Debugger 不只用於除錯或找出潛在的軟體缺陷，我們面對的是一系列從未參與設計開發的軟體元件的集合，不僅對內部設計一無所知，更別論整體之工作流程，這時候理解系統運作的機制，除了多如牛毛的原始程式碼外，就是加入合適的偵錯訊息，用單步執行(single-stepping)或在可疑處加入中斷點(breakpoint)，並在執行時期(runtime)嘗試不同的組態設定與程式流程，端詳其細部變化。hack 一詞套用於軟體開發相當傳神，而成功的 hack 更應針對特定需求，在不需大幅更動系統架構的前提下，提出有效且清楚的修正與改進，具備高度彈性的 Debugger 在這時就是最佳的利器。

再者，對於學習電腦科學或者研究演算法的學生來說，擁有一個可清楚得知系統狀況、即時監試特定變數變化的 Debugger，則可大幅增進對演算法實際運作的印象，原本沉睡於紙本的演算法描述，頓時躍然紙上，可清楚得知程式運作原理，無論是反覆性(iterative)或遞迴性(recursive)的設計，都能全面觀察系統運作的機制。

作為一個 Debugger，GDB 具備其他開發工具產品所沒有的特色，首先是廣泛的平台支援，無論是行動通訊界常見的 Symbian<sup>[5]</sup>與 PalmOS<sup>[6]</sup>，抑或是老古董的 VAX 與 PDP-11 工作站<sup>[7]</sup>，GDB 都以一致的功能提供完整的系統偵錯。GDB 的技術與實做完全開放，不壟斷於任何一家大廠，除了自由軟體社群快速且跨國際的支援與互助合作外，全球有若干家以 GNU Toolchain 為核心技術的專業公司，如 MontaVista Software<sup>[8]</sup>、RedHat/Cygnus<sup>[9]</sup>、Wasabi Systems<sup>[10]</sup>、CodeSourcery<sup>[11]</sup>等，而處理器晶片大廠，如 Intel、ARM，與 MIPS 等，都有全職工程人員投入 GNU 開發工具(含 GDB)的內部開發與平台支援，與「淪為廢墟的神殿」<sup>[12]</sup>有著本質上的差異。

與其他 GNU 開發工具一般，GDB 的核心設計理念之一就是可攜性(portability)，透過組態的更改，很快就可調整為特定之硬體架構。最特別之處則在於其 simulation target 設計，顧名思義就是內建模擬特定硬體與指令集的機制，可用以作移植與驗證新硬體平台，甚至包含設計中的硬體架構，達到 hardware-software co-design 的前期開發準備。GDB 這項機制的應用不限於作指令集的模擬，許多研究單位陸續以此為基礎，發展許多創新的發展，如普林斯頓大學的 EMSIM (Embedded StrongARM Energy Simulator)<sup>[13]</sup>將 StrongARM 平台的 GDB Simulation 加入對能源管理的模擬，可作為 Embedded Linux 實驗階段的能源分析與量化，中國北京清華大學的團隊更是大幅改進既有機制，對泛 ARM 架構的硬體與週邊，做出 SkyEye<sup>[14]</sup>這個特別的系統模擬器，在嵌入式系統應用常見的 LCD、Touch screen、Flash，與網路晶片等，做了近似的模擬，完成度相當高。

所以，一言以蔽之，為何要學習並精通 GDB 呢？不只要除錯，更要理解系統運作的機制。

## 參考資料

- [1] GNU Compiler Collections,  
<http://gcc.gnu.org/>
- [2] The Mythical Man—Month: Essays on Software Engineering,  
[http://en.wikipedia.org/wiki/The\\_Mythical\\_Man—Month](http://en.wikipedia.org/wiki/The_Mythical_Man—Month)
- [3] GDB: The GNU Project Debugger,  
<http://www.gnu.org/software/gdb/>
- [4] Google 時代的程式撰寫,  
[http://www.javaworld.com.tw/roller/page/qing?entry=2006\\_4\\_18\\_Google\\_Programming](http://www.javaworld.com.tw/roller/page/qing?entry=2006_4_18_Google_Programming)
- [5] Symbian OS — the mobile operation system  
<http://www.symbian.com/>
- [6] Palm OS  
<http://www.palm.com/>
- [7] The history and legacy of the PDP—11 series of 16—bit minicomputers produced by Digital Equipment Corporation from 1970 to 1990,  
<http://www.pdp11.org/>
- [8] MontaVista Software  
<http://www.mvista.com/>
- [9] RedHat — The Open Source Leader  
<http://www.redhat.com/>
- [10] Wasabi Systems  
<http://www.wasabisystems.com/>
- [11] CodeSourcery  
<http://www.codesourcery.com/>
- [12] 「淪為廢墟的神殿」暗喻 Borland 以 Delphi 系列開發工具馳騁於 RAD (Rapid Application Development) 與 IDE (Integrated Development Environment) 領域的豐功偉業，而多年之後卻被迫售出的窘境，筆者曾撰寫一篇短文介紹此「神殿」：  
<http://blog.linux.org.tw/~jserv/archives/001516.html>
- [13] EMSIM—2.0: Embedded StrongARM Energy Simulator  
<http://www.princeton.edu/~cad/emsim/>
- [14] SkyEye — Open Source Simulator  
<http://www.skyeye.org/>