**專家心得報告之一**

106級甲班 F74024070 蔡侑軒

今日講師：黃敬群老師(Jserv)

上課時間：105年2月26日、3月4日 下午兩點到五點

簡介：

台灣苗栗人，現居台南市，在各大學任教

成功大學資訊工程系92級

英文名Jim Huang

經歷：

台達電子

聯發科技

華碩集團

Open source專案參與：

AOSP(Android Open Source Project)

新酷音輸入法

Kaffe Virtual Machine

GCC/GCJ(GCC for Java)

LXDE

第一講：Compiler的多元應用

◎不要小看口袋中的Android手機 (談手機內的Compiler)

※預設安裝在系統底層

* ART或 Dalvik
* ART：用於Android 5以後，在安裝應用程式時，直接把代碼“預編譯“成機器語言(即AOT [Ahead-of-Time] complier)。優點在於有效率、耗電少、占內存低；缺點是應用程式的存儲空間變大，且安裝時間變長(但瑕不掩瑜，因安裝只有一次)
* Dalvik：用於Android4.4以下，JIT compiler，Dalvik會向硬體“解釋“應用程式代碼。優點是能讓同一應用程式運行在不同硬體上，直接促成Android系統的普及；缺點是耗電快、占內存大、運行效率低，被稱為Android的“毒瘤“
* JavaScript：使用 v8 engine
* 在執行前將JavaScript直接編譯成機器碼以提高效能
* 內建 PCRE engine ([irregexp](https://github.com/v8/v8/blob/master/src/regexp-macro-assembler-irregexp-inl.h))，JIT compiler
* RenderScript
* 包含：通用計算API、類似於OpenCL的計算API、類似C99的手稿語言
* 3D圖形計算
* 使用LLVM，對編譯、鏈結、執行及閒置的優化
* FreeType：bytecode interpreter
* bytecode interpreter，提供簡單易用且統一的介面來存取字型檔案
* GLSL (OpenGL Shading Language)
* Based on C的著色語言
* libpixelflinger：用於軟體的 blender，搭配 SurfaceFlinger 和 libagl
* codeflinger 動態產生 ARM 與 MIPS 的機械碼
* 像素級別的基本處理
* SQLite：資料庫
* regex ([regcomp.c](https://android.googlesource.com/platform/bionic/+/master/libc/upstream-netbsd/lib/libc/regex/regcomp.c) 的 "emit")
* 處理Regular Expression，JIT compiler
* iproute2: 處理 bpf 的 bytecode

※Android 應用程式裡頭的動態編譯器：

* 若安裝以 Unity3D 開發的遊戲，那就會有 Mono (C# Runtime; JIT compiler)
* YouTube.apk 裡面有 luajit.so (JIT compiler)

◎Web引擎&前端技術

* Google Closure Compiler：產生最佳化(最小)的JavaScript程式碼，以提升網頁效能
* Web Assembly：用低階的程式語言來加速執行效率(多用C/C++)

◎字串和資料處理

* qrintf-libtooling：允許改寫為可由靜態操作格式的編譯器技巧，相比sprintf()可提升10倍效能
* Regex(regular expression)
* vpacker：用LLVM+JIT作整數壓縮

第二講：From Source to Binary – How A Compiler Works

* 程式執行流程
  1. 先呼叫fork+execve，讓kernel準備建立新的process
  2. 配置process空間，並載入所需之動態連結器(EX：ld-linux.so)、動態函式庫(EX：libc.so for c)
  3. libc.so的libc\_start\_main函式會跳入執行檔的進入點(main)
  4. 執行完畢後，呼叫exit，由kernel善後
* 編譯流程
  1. C preprocessor會展開標頭檔 (#include…)
  2. C program (hello.c) 經Compiler(cc 編譯器) 產生對應的Assembly file (hello.s)，函數僅填名稱
  3. Assembly file經Assembler (as 組譯器) 產生 Object file(hello.o) (機器碼)
  4. 再經Linker(ld 連結器) ，找到函數的位址填入，形成 hello執行檔

※GCC是Compiler Driver，非Compiler

※程式語言不是早出現的就比較低階

* 編譯器術語
* Basic Block：單一進入點、單一出口點的程式區段
* CFG(Control Flow Graph)：程式的流程圖，用箭頭連結Basic Block
* From 原始碼 To 目標語言

1. 做字彙分析(正規語言)
2. 語法分析，形成語法樹
3. 語意分析(Type checking…)
4. 最佳化而成目標語言

* IR(Intermediate Representation)
* 引入IR，增加前後端彈性
* 最佳化方法
* Code Motion(程式碼搬移)：有些運算提前執行會比較快(EX：Loop invariant)
* Dead Code Elimination：刪除不需要的Code
  + 問題：Pointer Aliasing(不同指標指向同一記憶體)
  + 解決：設不同type的pointer沒有aliasing
* SSA(Static Single Assignment) Form：每個被assign的變數會有不同的version number
  + Constant Propagation(常數傳遞)
* IPO(inter-Procedural Optimization)
  + function inlining是其一，即將function body copy到caller上

* GCC前端：Source => AST(parsing tree + 語意) => Generic
* GCC中端：Gimple & Tree SSA Optimizer，Gimple可被化簡成SSA Form
* GCC後端：Register Transfer Language(RTL)，類Lisp語法，用virtual register(無限register)，Peephole optimization(即只考慮附近指令的最佳化)
  + 暫存器分配(Register Allocation)：著色問題
  + 管線排程
* Static Global Variable(只有這個Compilation Unit看得到)，編譯器可砍
* non-Static Global Variable(所有Compilation Unit看得到)，不可砍
* 為求最佳化，GCC會將標準函式轉為其他合適方式處理(printf=>puts、putc)

課程心得：

我覺得Compiler是一個很有趣的東西，從最一開始它被推出來只是為了當作人寫程式(代碼)與機器之間的溝通橋樑，到現在隨著時代進步，它有著越來越多且越強大的最佳化能力，就好像我們初學coding，總是先求「有」再求「好」一樣!但有趣的地方是，我們希望隨著時間與自己技術能力的進步，可以把code寫得越短越精煉越好，但Compiler卻不同!現今所求的Complier，似乎不以把code縮在萬行為目標，反之卻是不斷地增加code來擴充Compiler的功能!這點令我非常起疑：「在Compiler中把這些code做掉跟在執行時做會有差別嘛!? 這些最佳化不就只是把執行時的code搬到Compile時做掉而後再把結果貼到執行code上? 這一搬一放中間的overhead難道不會使得程式在執行時更慢嘛?」特別是聽到SSA Form時我更是這麼想。直到第二次下課，我跑去想問老師這問題時才知道，因為Compile這事基本上只會做一次，但從其產生出來的執行檔卻是有可能要重複跑千千萬萬次!也就是說，即便Complier會因為其內多加了SSA Form而變慢十秒，執行檔也只加快一秒的時間，但只要這個執行檔會跑上百次，這Compiler多出的overhead就不這麼重要了!而另一點我所想到的，老師所沒提到的是，就如同在合作開發專題時，我們會盡可能的把複雜的計算(演算法)包成一個簡單明瞭的function供其他人做使用一樣，Compiler是不是也力求這樣的簡單結果呢?在以前，工程師們會花不少的時間在增進、縮短、加速自己所寫的原始code，也就是所謂的「最佳化」，這是十分需要雄厚的背景知識與經驗，但是在現代這追求「人人都可以寫code」的大千世界中，大家或許有能力把自己所需要的程式給寫出來，但卻沒辦法做到「最佳化」的動作，這時若可以讓Compiler內也有最佳化的功能，那至少能初步的改善效能，從而降低學習程式所要跨過的門檻吧?

但我覺得Jserv老師帶給我的，其實更多是關於未來的看法。他說他以前僅僅是一個苗栗鄕村的小孩，藉由電腦、coding，讓他一躍成為台灣甚至世界上舉足輕重的人。而且雖然有些半開玩笑的性質，但他不斷的鼓勵我們要拿問題去問接下來的講者，我想是想讓我們知道，「機會」這個東西，它不是那麼遙不可及的星辰，更不是千年一次的奇蹟，其實它一直都在我們身邊，只是我們「不知道」它在，甚至「不知道」原來它就是「機會」本身，由其在這科技發達的21世紀，其實「機會」它就沉在這汪洋的網路大海中等著被人發掘吧!