GCC 與系統程式學習筆記

GCC 與編譯流程

GCC 不僅是編譯器，而是一整套工具。

編譯 C 程式的常用選項：

- -c：只編譯，不連結，產生目的檔 (.o)。

- -S：產生組合語言檔 (.s)，S 要大寫。

- -g：包含除錯資訊，用於 GDB 等除錯工具。

- -o <file>：指定輸出檔名。

- 標準指定：-std=c99 指定 C99 標準。

- 優化等級：-O0 (最低優化), -O3 (最高優化)。

- 在 Mac/Clang 環境下，可能需要特定的 flag 如 -fno-integrated-as 才能看到類似 Linux GCC 的組合語言輸出。

編譯過程通常包括：預處理 (preprocessor)、編譯 (compilation)、組合 (assembly)、連結 (linking)。gcc 指令可以自動完成整個流程。

目的檔 (.o) 是編譯後的二進位檔案，但尚未連結，不能直接執行。連結器 (linker, L) 負責將目的檔與函式庫連結成可執行檔。

Objdump 工具可以反組譯 (disassemble) 目的檔或可執行檔，顯示機器碼和對應的組合語言指令 (objdump -d <file>)。在某些平台或使用特定選項時，Objdump 可以顯示變數名等更多資訊 (如 objdump --prefix-addresses -S <file>)。

Make 工具 (Makefile) 用於自動化編譯過程，根據檔案的依賴關係決定哪些部分需要重新編譯。Makefile 中定義了目標 (targets) 和依賴 (dependencies)，以及生成目標的規則。常見目標包括 all (建置所有) 和 clean (清除生成檔案)。

GDB (GNU Debugger) / LLDB (Mac 上的除錯工具) 用於程式除錯。基本指令包括 b <location> (設定中斷點 breakpoint), r (執行 run), n (執行下一行 next), c (繼續執行 continue), p <variable> (印出變數值 print)。除錯資訊 (-g flag) 對除錯很有幫助。

編譯器與文法理論

編譯器的理論基礎之一是 生成語法 (Generative Grammar)，由 Chomsky 在 1957 年提出。

BNF (Backus-Naur Form) 是一種用於描述文法的符號系統，常見於上下文無關文法 (Context-Free Grammar)。編譯器基於上下文無關文法來描述程式語言的語法。

編譯器的第一階段通常是 剖析 (Parsing)，將符合文法的程式碼轉換為結構化的表示 (如語法樹)。剖析可以視為生成語法的逆過程。

課程中介紹了一個簡單的 C 語言編譯器 c4，用於理解編譯器的工作原理。c4 程式碼相對精簡。

c4 編譯器涉及詞法分析 (掃描原始碼產生 token) 和語法分析 (根據文法規則解析 token 序列)。

虛擬機與組合語言

虛擬機 (Virtual Machine, VM) 是一種經典的系統軟體。VM 有自己的指令集和暫存器。

VM 的核心暫存器通常包括：

- PC (Program Counter): 指向下一條要執行的指令的位址。

- BP (Base Pointer) / FP (Frame Pointer): 函式呼叫時的基底指標，用於定位區域變數和參數。

- SP (Stack Pointer): 指向堆疊頂端的位址。

- A (Accumulator): 累加器，許多運算預設在此暫存器上完成 (累加器架構)。

組合語言 (Assembly Language) 是機器碼的符號表示，與特定的 CPU 架構緊密相關。不同的 CPU (如 x86, ARM, RISC-V) 有不同的組合語言。

組合語言指令直接操作暫存器和記憶體。例如 mov (移動資料), add (相加), jmp (無條件跳躍), call (呼叫副程式), ret (從副程式返回)。

函式呼叫 在組合語言層級涉及複雜的步驟，包括：

- 將參數依呼叫約定 (calling convention) 推入堆疊或放入特定暫存器。

- 將返回位址 (return address) 推入堆疊。

- 保存呼叫者的 BP/FP 到堆疊，並設定新的 BP 指向當前函式堆疊框架的基底。

- 為區域變數在堆疊上分配空間。

- 執行函式體內的指令。

- 恢復呼叫者的 BP/FP。

- 清理參數佔用的堆疊空間 (依呼叫約定)。

- 跳轉到返回位址。

堆疊 (Stack) 用於儲存函式參數、返回位址、區域變數和被保存的暫存器。它是一種 LIFO (Last-In, First-Out) 的資料結構。

上下文切換 (Context Switch) 是作業系統在單一 CPU 核心上實現多工或多執行緒的機制。它通過保存當前執行單元 (行程或執行緒) 的狀態 (所有暫存器的值) 到記憶體，然後載入下一個執行單元的狀態來實現。頻繁的上下文切換會降低效能。