組合語言與系統程式

期末專題書面報告

第26組
資工2B 102502559 吳承霖

程式說明

這個部分我將會分為好幾個小節逐一解釋我的專題跟別人的不同之處，以及我為什麼想要用這麼與眾不同的方式完成我的Project，希望看過之後，能對交上的程式碼有更多了解。

1. It is GNU As , Not MASM

首先就是先說明我的專題不是用本學期老師教的MASM，而是用Linux平台流行的GNU Assembler(簡稱為GAS)撰寫，會用這個語言的原因，主要是因為之前為了研究，而使用GCC編譯器把C語言編譯成GAS，藉由對照研究程式碼，來了解程式語言每個動作底下是如何被實作，也因此了解如何用GAS呼叫C的Library(其實就跟使用MASM 呼叫 Irvine的procedure的動作一樣)，也對GAS的結構和一些Directive有更多了解。其次的原因，是因為手邊沒有windows環境..所以就用我熟悉的GAS在Linux環境上開發了。

2. GTK

本次專題的成品不僅以視窗的介面呈現，還包括在視窗上繪圖以及動畫的效果，至於大部分的繪圖是另外Link一個我用C語言寫的小程式，至於組合語言控制的部分，就是遊戲的各項參數計算，還有偵測和處理各個事件的觸發，其中也使用了很多Gtk Library的function。

3. GAS沒提供Array和Struct？那就自己算位址吧

GAS中似乎沒有提供像MASM裡面的struct以及array那樣好用的功能，不過既然知道了程式語言們是怎麼存取Array和struct的每一個元素，那就用最原使的方式吧：自己切一塊區域存各個值，然後透過Base+offset的方式存取到各個element/attribute。下圖的程式碼片段是我透過這個方式打算存取一個struct中的某個array，在C語言的概念就是存取『第*current_level*個map中的第*source*個block』

```
133 .LOOP0B:
134     mov eax, current_level
135     movsx rax, eax
136     imul rax, rax, 316              # current_level*316
137     mov ecx, DWORD PTR [rbp-20]     # source
138     movsx rcx, ecx
139     sal rcx, 4
140     add rax, rcx                    # current_level*316+source*16
141     # maps[current_level].blocks[source]
```

解釋一下上圖的程式碼，首先要先找到名為第current_level個map，而已知每一個map的大小為316 Bytes(算出來的)，所以maps[current_level]的offset就可以用『current_level*316』算出來，接下來要再加上第source個block的offset，已知每一個block大小為16 Bytes，所以maps[current_level].block[source]的offset就是『current_level*316+source*16』了

備註: GAS中沒有Procedure內設置LOCAL變數的功能，所以在Procedure中設置變數或存放參數都是透過rbp+offset存取Stack中該變數。

4.如何用組合語言呼叫C library和GTK？

其實呼叫C語言的function或是GTK Library的function，道理就如同這學期上機或是作業用Masm呼叫Irvine的Procedure一樣，差別是統一都是用call而沒有INVOKE如此方便的功能，所有參數的傳遞都是在呼叫前把資料先挪到如ESI或EDI此類的暫存器，於是我在設計自訂的procedure的時候也參考了這個模式，procedure內一開始會把特定暫存器內的值當成參數，然後放進Stack中存起來。

```
mov esi, OFFSET .MODE
mov edi, OFFSET .FILE
call    fopen
mov QWORD PTR [rbp-8], rax
```

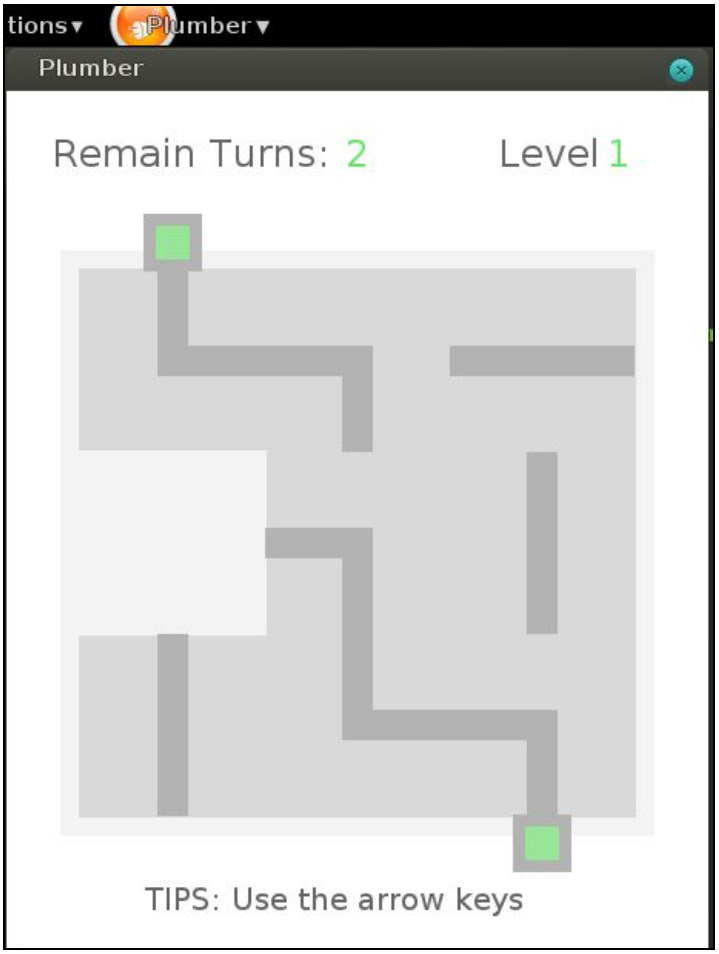
上圖即是一個以組合語言呼叫C語言fopen(filename, open_mode)的範例，在專案中我透過讀檔來把遊戲地圖的各個參數和資料讀進來，這樣只要編輯那份文字檔，就可以新增或修改遊戲關卡了。程式碼中可以看到fopen所要求作為參數傳遞的暫存器為esi以及edi，於是就把檔案名稱的字串以及開檔模式的字串存入暫存器中，然後再呼叫fopen。而通常，如果該procedure有回傳值，都會放置於eax/rax暫存器中，fopen()會回傳開啟的檔案的addr，存在rax中，所以呼叫完要拿到回傳值，把rax的值宜出來置入Stack即可使用。

附註：編譯環境

這份期末專題我是在Ubuntu 14.04上撰寫，所相依的package有gcc以及libgtk-3-dev，若是使用debain系列的作業系統，可以在指令列打 `sh configure.sh` 檢查是否有這些套件，若已經安裝這些相依套件，便可以在命令列數入 `./Makefile`，即可把原始碼編譯成執行檔。

因為編譯好的執行檔無法跨平台，所以就以提供編譯檔麻煩助教打以上指令來取得執行檔了，不好意思

程式結果截圖



圖中灰色區域左方有一個空白缺塊，玩家利用上下左右鍵控制此空白格與上/下/左/右的格子互換位置，在有限的移動次數內讓兩個綠點之間的線條能夠連接起來。

連接完成後即為通過此關卡，按下空白鍵後可以跳至下一關。

心得感想

這次的期末專題不同於前幾次的作業和上機實習，沒有老師的講解、沒有投影片介紹Procedure用法，所擁有的只能在網路上寥寥無幾的資料中尋找寶藏。

不過靠這樣的學習方式竟然讓我創造出許多屬於我自己的寶藏，成品是其次，如果不用組合語言寫而是用其他高等方便的語言寫或許幾個小時就寫好了，但因為組合語言低階到直接一對一映射到機器碼，讓我了解了很多平常我們習以為常的高階語言的動作，抽象背後的實作。

更了解程式語言的運作，獲得這樣的認知究竟對我有什麼幫助呢？我想就是因為了解輕鬆的一行高階語言，竟然可能會讓硬體做許多無謂的浪費，所以每一行都會思考是否最有效率。

最後，其實到現在還是很擔心如果不是用MASM的期末專題，老師和助教能不能接受，不過我所使用的方法其實讓我做專題的時間比別的組多得許多，並不是為了偷懶所選擇的捷徑。

組合語言那低階到直接了當的精神和知識，我已經透過這次的期末專題，獲益良多，也很喜歡在撰寫的過程中參雜成就感與探索未知的感覺。