

Project #3. Multitasking

在現今的電腦系統中，搭配的多半是擁有多個核心的 CPU，意味著電腦可以將不同的工作分配給不同的核心，讓系統能夠同時執行多個應用，以提升系統整體運作的效能。

在撰寫程式時，我們會希望能夠將 CPU 的使用率提升到最高，也就是最大化的去利用 CPU 的資源，因此，我們會利用「Multitasking」的技術，讓程式在執行的過程中，如果碰到需要等待 I/O 結束的情況，也就是不會使用到程式所佔用的 CPU 的資源的時候，不會讓 CPU 閒置，而是會將 CPU 的資源用來執行程式的其他部分，目前常見的 Multitasking 的技術則分成「thread-based」和「event-driven」兩種情況，以下會透過一些簡單的範例程式，來說明這兩種不同的情況。

1. Thread-based[1]

```
import threading

# Our thread class
class MyThread (threading.Thread):

    def __init__(self,x):
        self.__x = x
        threading.Thread.__init__(self)

    def run (self):
        print str(self.__x)

# Start 10 threads.
for x in xrange(10):
    MyThread(x).start()
```

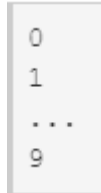
圖表 1. 透過 python3 所撰寫的 thread 範例程式

以上是一個透過 python3 所撰寫的 thread 的範例程式，在 python3 中如果要使用 thread，必須得倚賴 threading 這個 module 來達成，因此，我們先在程式中 import 該 module，並定義一個名為 MyThread 的 class，讓其繼承 Threading.Thread 這個 class，接著再定義以下兩個函數。

首先是「__init__()」這個函數，這是用來進行一些初始化的動作的函數，以此範例程式而言，在此函數中，我們一共會傳入兩個參數，分別是「self」和「x」，「self」指的是當你 create 此 class 的 instance 的時候，會指向 instance 本身，「x」則是我們在 create instance 時，額外傳入的另一個參數，在「__init__()」之中，我們先將傳入的 x 存進「self.__x」這個屬性之中，再呼叫 parent class 的「__init__()」函數，來完成剩餘的初始化動作。

再來，我們還要定義「run()」這個函數，這個函數是當 MyThread 這個 class 的 instance 的「start()」被觸發，真正開始執行此 thread 的時候會做的動作，在此範例程式中，「run()」所傳入的參數只有「self」，做的事情也很簡單，就是將在執行「__init__()」時所儲存的「self.__x」給印出來，此 thread 就結束了。

範例程式中最下面的部分，則是透過 for 迴圈，去產生 10 個 thread，並傳入從「0~9」這些不同的參數，若是執行此程式，則應該會得到如下圖的結果。



```
0
1
...
9
```

圖表 2. Thread 範例程式的執行結果

2. Event-driven

```
1 var net = require('net');
2
3 var server = net.createServer(function(socket) {
4   socket.on('data', function(data) {
5     console.log(data.toString());
6   });
7 });
8
9 server.listen(3000, '127.0.0.1');
```

圖表 3. 透過 NodeJS 所撰寫的 event-driven 的範例程式

以上是使用 NodeJS 所撰寫的一個簡單的 TCP Server 的範例程式，我們先引入建立 TCP Server 所必要的「net」這個 module，接著透過該 module 的「createServer()」來創建一個 TCP Server，當有人連進這個 TCP Server 時，會觸發我們先前 call「createServer()」時，所傳入的這個 callback 函式，這個 callback 函式裡的「socket」參數，代表的就是所建立的這條 TCP 連線，其中定義了當有人透過這個 socket 將資料傳進來，也就是觸發了「data」這個 event 時，會呼叫後面的 callback 函式，其中的「data」參數代表的就是所傳入的資料，我們這裡做的事情也很簡單，就只是將收到的資料直接印出來而已。

最下面的那一行，則是讓 TCP Server listen 在 localhost 的 3000 port 上，這樣 TCP Client 才可以透過該 port 對此 TCP Server 進行連線。

作業內容

在此次 Project 之中，我們希望大家可以透過撰寫一支簡單的 TCP Server 程式，來理解到如何使用 Multitasking 的技術，並撰寫一篇 report，說明一下你所撰寫的程式的內容，過程中遇到的問題，以及你是如何解決的，並做一張表格以顯示你所撰寫的 Server 的效能。

此次作業一共會分成兩大部分，說明如下：

1. TCP Client

此部分的程式不需要自行撰寫，助教會提供一支由 NodeJS 所撰寫的 TCP Client 程式，其執行後的步驟如下：

Step1. 取得系統中的 CPU 數量，並 fork 出相同數量的 process

Step2. 每一支 process 會和 Server 建立對應數量的連線

Step3. 每一條連線建立成功後，會產生一組 string 亂數並丟給 Server，Server 進行 PoW(Proof of Work)的運算後，會回傳對應的資料

Step4. 當 Client 收到資料時，會隨機產生一個 int 亂數，若亂數符合特定的條件，則進行抽檢，以確認 Server 所回傳的資料是否符合 PoW 的要求，若不需抽檢，則將 response 數+1，並回到 Step3 繼續執行，但不須重新建立連線

Step5. 若抽檢成功，則將 response 數+1，並回到 Step3 繼續執行，但不須重新建立連線，若抽檢失敗，則 TCP Client 會印出錯誤訊息，並結束程式

Step6. 程式執行一定的時間後，fork 出來的 process 會結束，並將最後的統計結果(總回覆數、測試時間、PoW 的難度、回覆數/秒)印在 console 上以便查看

2. TCP Server

大家可以使用自己所熟悉的程式語言，去撰寫一支相對應的 TCP Server 程式，其執行後的步驟應如下：

Step1. 此 TCP Server 會 listen 在 localhost 的 port 20000，等待來自 TCP Client 的連線，要特別注意的是，Server 要盡量能同時處理多個 Client 的連線請求

Step2. 連線成功後，會收到來自 Client 所隨機出來的一組 string 亂數，你必須要把該 string 亂數做 PoW 的運算，得到需要的結果之後，將收到的亂數、seed 以及 hash 結果一起回傳給 Client，並等著處理下一次 Client 所丟過來的訊息

作業繳交

將「你所撰寫的 TCP Server 的程式碼」、「report」壓縮成一個「學號_姓名.zip」的檔案，並在繳交期限內上傳到 e3 上。

評分方式

1. TCP Server 能否正常運作 (60%)
2. report 內容 (30%)
3. TCP Server 的效能 (10%)

備註

1. Proof of Work 步驟說明如下：

Step1. 產生一組亂數，做為 hash 時所使用的 seed

Step2. 將 seed 和收到的亂數 string 串起來，也就是「seed+亂數 string」

Step3. 使用 sha256 這種 hash 函數來計算 hash

Step4. 如果 hash 後的結果符合事先設定的 PoW 的難度(hash 後得到的結果的開頭要有特定數量的 0)，則將結果回傳給 Server，否則，回到 Step1 繼續執行

2. 在傳送訊息的時候，要特別注意要在訊息的最後加上一個換行符號「\n」以做為不同條訊息之間的分隔符號

3. 在收到訊息的時候，要先將訊息結尾的換行符號給移除，再利用換行符號「\n」來將不同條的訊息給分開來，再利用 forEach 之類的函式來對每一條訊息個別進行處理

4. Server 所收到的隨機亂數，會是一組由 5 個 char 所組成的隨機 string，例如：「70NVO」

5. Server 回傳給 Client 的資料，會是由「message」、「seed」和「hash value」所組成的，「message」是從 Client 所收到的隨機 string，「seed」也是由 5 個 char 所組成的隨機 string，「hash value」則是將「seed」和「message」串起來後，透過 sha256 所 hash 出來的結果，這三者是一起傳送的，但會用「,」來做為區隔，例如：
「70NVO,oH2eX,000373f186fcd7c03f1f2887cf9aff20a2ca7ee14a44dd3e59653f636375df00」

6. 在執行 Client 之前，需要事先安裝「NodeJS」和「npm」，後者是 NodeJS 的套件管理系統，需要用其安裝「js-sha256」這個套件之後才能執行

7. 在 TCP Client 之中，有幾個可以調控的變數，分別說明如下：

A. connectionNumber：Client 連線到 Server 的連線總數，會平均分配給各個 fork 出來的 process

B. testTime：此次測試的總時間長短

C. PoWDifficulty：PoW 的難度，若為 3，則代表 Server 所回傳的 hash 必須至少要有 3 個 0 做為開頭

8. 在 TCP Client 之中，已將一些 debug 訊息給註解起來，若有需要，可自行取消註解以方便測試

9. 每次測試前，要確定 Server 和 Client 所設定的 PoW 難度是相同的

10. 要根據不同的 PoW 難度來做測試，並將結果設計成如下圖的表格，在相同的 PoW 難度之下，若每秒的回覆數越高，則代表 Server 的效能越好

PoW 難度	0	1	2	3	4	5	6
回覆數/秒	119873	27475	2700	177	11	0.8	0.033

11. 根據你所使用的 hash 函數的效率，有可能會影響到 Server 的效能

12. 若是透過 python 來撰寫具有 thread 功能的 TCP Server，可參考以下資料：
https://python3-cookbook.readthedocs.io/zh_CN/latest/c11/p02_creating_tcp_server.html
13. 若是透過 NodeJS，使用 event-driven 的方式來撰寫 TCP Server，可參考以下資料：
<https://bonze.tw/%E4%BD%BF%E7%94%A8-node-js-%E5%BB%BA%E6%A7%8B-tcp-server/>

參考資料

- [1] <http://tw.gitbook.net/t/python3/article-83.html>