OS HW1 0711529 陳冠儒

python 版本：3.8.5

input data：助教提供之範例

我們先來討論一下Multi-threading跟Multi-processing的差異：

Process就好比是一間工廠，而thread則是員工，如果電腦擁有多個處理核心，就代表系統可以同時調用的員工數量增加，此時可以有兩種策略。

1. Multi-processing：同一時間內為各家工廠都分配一個員工去作事， Multi-processing 平行執行。跟single process處理比起來，其優點在於可以在**相同的時間內完成較多的工作**。
2. Multi-threading：在同一時間內把所有員工都派到同一家工廠去工作，Multi-threading平行執行。相較於single threading處理方式，它**有機會讓相同的工作在比較短的時間內完成**。

Task的設計：

* Task1：Proof of Work
  + 使用套件：hashlib
  + 方式：利用遍歷從最小一直試到最大，直到找到結果就return
* Task2：Get Url Header
  + 使用套件：BeatutifulSoup的bs4、requests
  + 方式：requests.get(url)得到html後，再用bs4去得到head.title.text

1. 執行緒數量對效能的影響

* Multi-threading的設計
* 使用套件：threading、queue
* 方式：將所有task都先放到queue中，再把他們get出來分給各個thread去執行
* 執行結果
* Task 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| threads number | 1 | 2 | 4 | 100 |
| time1(s) | 122.24437 | 124.57429 | 122.61275 | 125.53242 |
| time2(s) | 121.93974 | 121.60334 | 122.87616 | 125.94289 |
| time3(s) | 120.19224 | 122.06673 | 122.02587 | 125.99407 |
| average time(s) | 121.45878 | 122.74812 | 122.50493 | 125.82313 |

* Task 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| threads number | 1 | 2 | 4 | 100 |
| time1(s) | 91.484702 | 13.94796 | 11.97931 | 6.51481 |
| time2(s) | 87.942572 | 26.50696 | 10.80837 | 6.4377 |
| time3(s) | 54.606038 | 20.80738 | 10.64665 | 6.579089 |
| average time(s) | 78.011104 | 20.42077 | 11.14478 | 6.510533 |

* 結果說明

Multi-thread的優點是

1. 在一部分blocked時，可以切換到另一部份繼續跑

2. 同一個process下的thread可以共享資源

3. 他進行context switch的速度遠快於process

但我們可以發現在task1下他的速度沒有下降，因為task1是CPU bound，需要進行大量的計算，並且沒有任何I/O burst，所以此時的thread並沒有任何作用，甚至可能會因為要create thread而使得花費的時間更久，以上面工廠的例子來說明，就是一個工作已經使得一間工廠的運作達到了最大，故此時即使有再多的員工也無法增快速度。

在task2下，在requests.get(url)時會被blocked住，所以此時可以發揮thread的效用，進行concurrent的運行，使得花費時間大幅的下降。

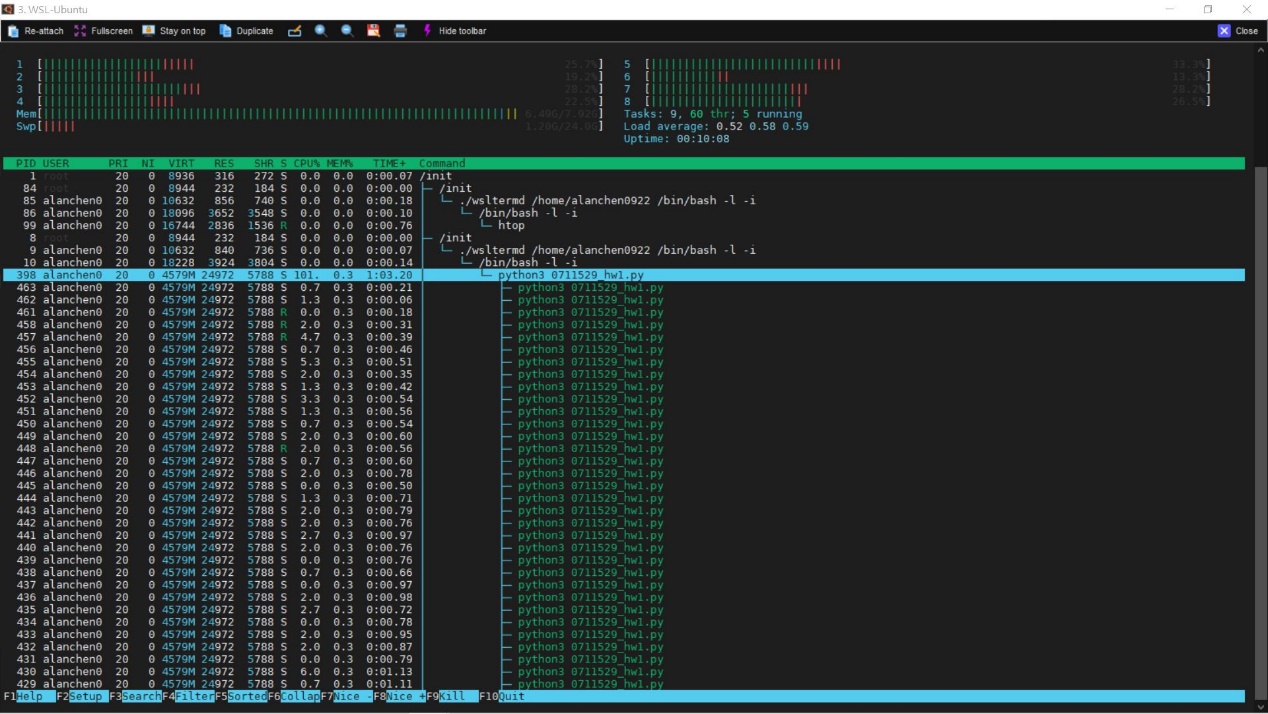
1. 行程數對效能的影響

* Multi-Process設計
  + 使用套件：multiprocessing的Pool
  + 方式：利用processing pool將task map到各個process去執行。
* 執行結果
* Task 1

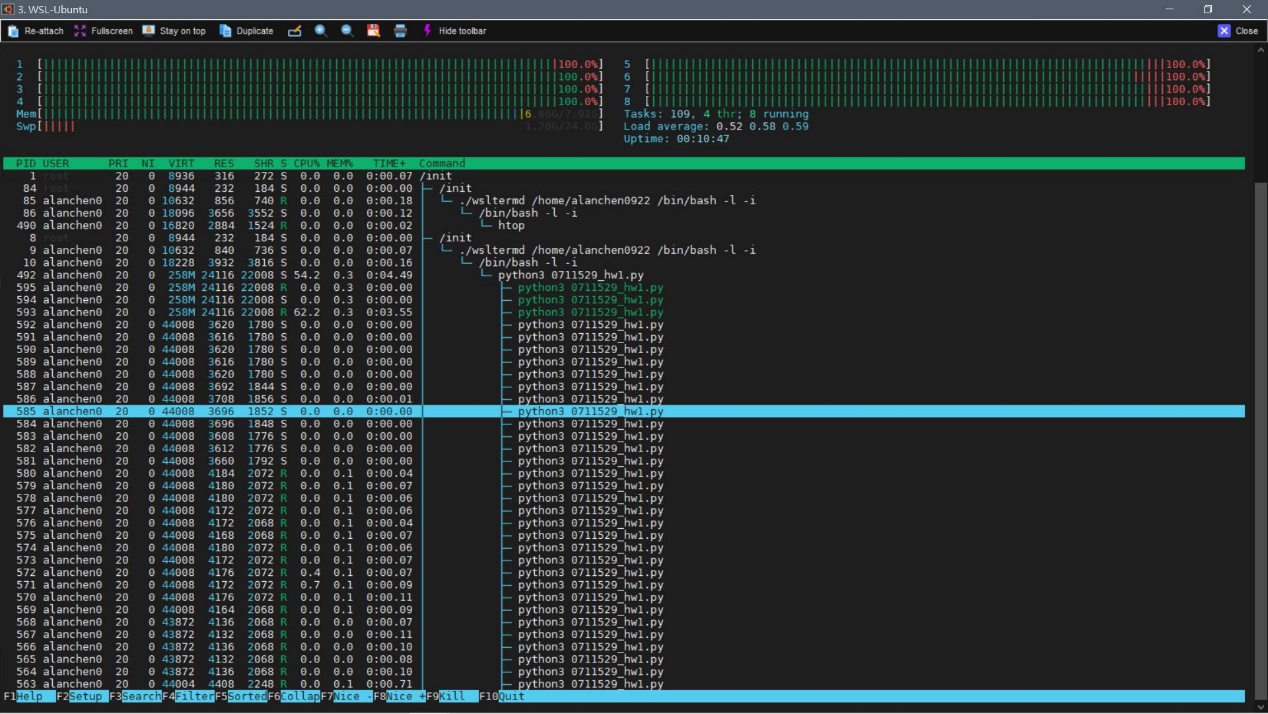
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| processes number | 1 | 2 | 4 | 100 |
| time1(s) | 137.23133 | 85.092686 | 62.814119 | 47.427244 |
| time2(s) | 133.15478 | 87.785173 | 62.830519 | 49.103679 |
| time3(s) | 133.69309 | 86.995256 | 62.873352 | 50.542476 |
| average time(s) | 134.69306 | 86.624372 | 62.83933 | 49.024466 |

* Task 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| processes number | 1 | 2 | 4 | 100 |
| time1(s) | 99.218601 | 21.00603 | 16.07411 | 5.701726 |
| time2(s) | 67.952653 | 18.74726 | 9.836351 | 5.54199 |
| time3(s) | 89.621445 | 20.70694 | 14.57304 | 4.120386 |
| average time(s) | 85.597566 | 20.15341 | 13.4945 | 5.12136733 |



Multi-thread Task1 100 thread



Multi-process Task1 100 process

* 結果說明

由上面兩個執行狀況可以看到multi-process可以使用全部的CPU資源，而multi-thread只能使用一個process能使用的CPU資源，故multi-process在task1和task2執行的速度都能隨著process的增加而減少。

1. 多執行緒、多行程、協程的效能比較

* Corotine設計
  + 使用套件：asyncio
  + 方式：使用get\_event\_loop()並create\_task()將各個task包裝進去，最後在requests.get設下await run\_in\_executor讓他在那邊中斷與切換。
* 執行結果
* Task 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 單執行緒 | 100 個執行緒 | 100 個行程 | 協程 |
| time1 | 122.24437 | 125.53242 | 47.427244 | 124.19102 |
| time2 | 121.93974 | 125.94289 | 49.103679 | 120.61525 |
| time3 | 120.19224 | 125.99407 | 50.542476 | 128.11047 |
| average time | 121.45878 | 125.82313 | 49.024466 | 124.30558 |

* Task 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 單執行緒 | 100 個執行緒 | 100 個行程 | 協程 |
| time1 | 91.484702 | 6.51481 | 5.701726 | 6.538094 |
| time2 | 87.942572 | 6.4377 | 5.54199 | 6.858862 |
| time3 | 54.606038 | 6.579089 | 4.120386 | 6.601222 |
| average time | 78.011104 | 6.510533 | 5.121367 | 6.666059 |

* 結果說明

首先先來說一下coroutine的原理



Coroutine 是輕量化的 Thread (Light-weight Thread)，他是屬於協同式多工，程式會定時放棄已佔有的執行資源讓其他程式執行，由程式自己讓出執行資源，作業系統不會干涉。而multi-thread則是搶佔式多工，作業系統會根據程式的優先權去安排當下哪個程式能有資源去執行。因為coroutine之間的切換是在上層，不需要由OS來處理，故他的context switch負擔會較mutli-thread更小。

由結果可以發現，task1的時間並沒有縮減，因為如果在multi-thread那裏說的，task1是CPU bound，並且完全沒有I/O burst，所以coroutine的結果跟multi-thread相似沒有什麼效果，而multi-processing則明顯的可以縮短時間，利用CPU資源去縮短運算時間。

而在task2在requests.get(url)的地方會有blocked，此時coroutine就可以切換到其他的去執行，而不用等他，故可以減少執行的時間。而結果顯示multi-thread和coroutine的時間會比較接近，因為他們的原理比較相同，都是在同一個process下去進行切換，而multi-process的速度還是最快，因為他能夠使用的資源會是最多的，在不會有太大的context switch overhead下，他會是最快的。

四、Reference:

* coroutine原理：<https://ithelp.ithome.com.tw/articles/10235352?sc=iThomeR>
* coroutine python：<https://docs.python.org/zh-cn/3/library/asyncio-task.html>