**# OS Project 1 Report**

**資工二 謝心默 B06505017**

**Design**

**General**

- Main process和 child 分別使用兩顆CPU來跑以防止互相搶資源

- 在child的那顆CPU上，只有當下正在執行的process會有較高的priority

- 使用 share memory來

main process及之後fork的child們使用兩個不同的prioirty，只有當下正在執行的process使用較高的priority(因此我只使用一顆CPU)。 使用share memory來存取clock以及每個task的資訊(進來時間、剩餘執行時間)等需要分享的資訊，這樣每支process都能看到現在會不會有人要進來，如果有就(以調整priority的方式)將cpu歸還給main process，然後將這個時間點進來的task全部fork出來，再看是要交給上一位執行的process還是有人要preempt轉移cpu。

**FIFO**

First-Come, First-Served Scheduling

原理：

對於每個process，先來的先執行。

實作方式：

- 將所有process按照ready time由小到大排序

- 對於每個process，若其ready time為當前的時間，則將其標記為ready的狀態，當其ready time為當前ready狀態最小的process，執行直到結束

**RR**

原理

利用計時器固定週期，讓正在執行的process在執行一個週期後進入queue中，並從queue中選定下一個執行的process。

執行方式

- 將所有process按照ready time由小到大排序

- 對於每個process，若其ready time為當前的時間，則將該process放入queue中

- 如果目前沒有在執行process，並且queue中有process，則從queue中取出process執行，若在執行一個Time Quantum後沒有結束，則將其execution time減少一個週期，並將該process放回queue中

**SJF**

原理

選擇執行時間最短的process來執行，此處應用的SJF是non-preemptive的，不會中斷仍在執行的process

執行方式

- 將所有process照ready time由小到大排序，若ready time相同，再按照execution time由小到大排序

- 對於每個process，若其ready time為當前的時間，則將其標記為ready的狀態，當其為當前ready狀態最小execution time的process，執行直到結束

**PSJF**

原理

每次選擇剩餘執行時間最短的 process 來執行，當新產生的 process 執行時間更短時，就可以插隊（preemption）

執行方式

- 將所有process照ready time由小到大排序，若ready time相同，再按照execution time由小到大排序

- 對於每個process，若其ready time為當前的時間，則將其標記為ready的狀態，當其為當前ready狀態的process中最小execution time的process，則將原本在執行的process替換，並讓原本執行的process回到ready的狀態中。

- 與SJF相似，唯一不同點在於main process在fork新process後，如果新的process執行時間較短，則之後轉移cpu時，不是轉移回去，而是轉給新process

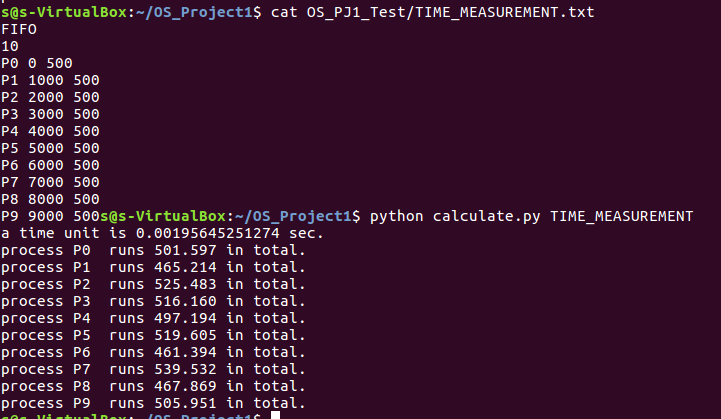
**核心版本**

環境:linux ubuntu 16.04 LTS Kernel:linux-4.14.25

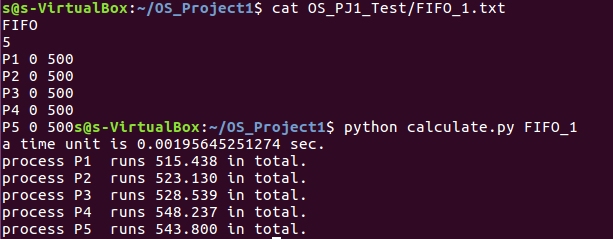
**比較**

使用程式(calculate.py)計算每個process用了幾個unit\_of\_time

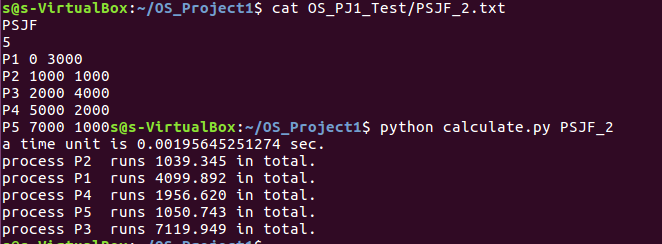
1. 首先看到TIME\_MEASUREMENT，每支process的範圍大概落在[500-40, 500+40]之間，也就是誤差在8%左右



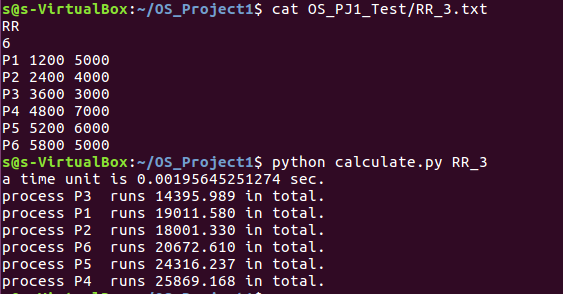
1. FIFO\_1的部分，誤差在+10%內



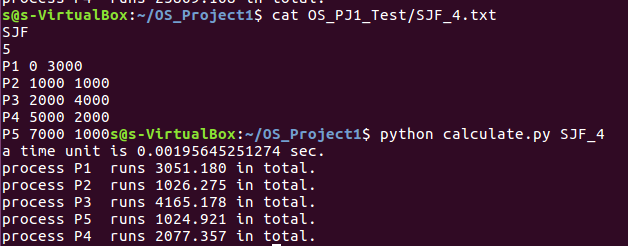
1. PSJF\_2的部分，因為有preempt，因此P1的總執行時間為4000，P2為1000，P3為7000，P4為2000，P5為1000，可看到誤差皆在5%內



1. RR\_3的部分，因為他們會一直切換，因此總執行時間比他們原定的執行時間增加了很多。理論值則應該各為:P1:18500，P2:17500，P3:14000，P4:25000，P5:23500，P6:20000，可以看到誤差都在5%內



1. SJF\_4的部分，因為沒有Preempt，因此誤差與要求的時間只在5%內



因此大概可以推論我的這份scheduler的誤差約會在5%內

**討論**

1. 在我的程式中，我是讓child來看是否有新的task出現，再喚醒scheduler去新增，但這部分應該是要交給scheduler來確認的，child不應該有辦法知道其他child的狀況。
2. 時間的定義上，在此使用unit of time，而我讓child也能更改目前的時間，但實際上會有專門的process來執行，且是用實際時間來看，會比較精準。
3. 不過因為這裡是跑空迴圈，但如果涉及到storage的操作，一個操作所需要使用的時間會大幅上升，因此若用幾個操作來看可能不太好，所以可能真的在做的時候可以用clock的值。
4. 我只有使用一顆CPU，因此main process在跑的時候可能會拖延到執行的時間，但因為操作數目不多，因此影響應該不大。main process會影響到只有在中途有新的task進來，或者是需要Preempt的時候，才會從child process退到main process。
5. 如果沒有涉及到其他process的情形，誤差的來源來自於非迴圈的操作，如if-else判斷是否該結束or換人等等

**備註**

* output資料夾底下的"xxx\_finish"紀錄執行該測資時，每個process開始(首次進入CPU)及結束的時間。