OS Project 1 Report

tags: `OS` `project` `OS project`

資工三 黃奕鈞 **B06902033**

設計

General

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。一開始按照 1. 進來時間 2. input 出現順序 的方式排序

FIFO

使用 Queue 來存取執行中的程式，如果做完就 DeQueue 然後回 main process，main process 如果看到 Queue 不是空的就作 Queue.front 的 process。在 EnQueue 的時候，因為排序的方式，故順序會正確，以此來達到 FIFO。

RR

與 FIFO 相同用 Queue 來實作。唯一不同是每支 Process 在執行的時候會記錄自己做了幾個回合(unit of time)了，如果大於零且為 500 的倍數就放在 Queue 的尾巴(EnQueue(Queue.front), DeQueue())。如此就能達到 RR 且新進來的 process 會進到 Queue 的尾巴了。

SJF

與上述兩者不同，在完成 child process 之後，main process 不是看 Queue 來找 process，而是看目前已經進來的 process 中，因此排序方式可與 FIFO 同，剩餘時間最短的(時間複雜度 O(N))。

PSJF

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。其他都與 SJF 相同。

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

```
s@s-VirtualBox:~/OS_Project1$ cat OS_PJ1_Test/TIME_MEASUREMENT.txt
FIFO
10
P0 0 500
P1 1000 500
P2 2000 500
P3 3000 500
P4 4000 500
P5 5000 500
P6 6000 500
P7 7000 500
P8 8000 500
P9 9000 500s@s-VirtualBox:~/OS_Project1$ python calculate.py TIME_MEASUREMENT
a time unit is 0.00195645251274 sec.
process P0  runs 501.597 in total.
process P1  runs 465.214 in total.
process P2  runs 525.483 in total.
process P3  runs 516.160 in total.
process P4  runs 497.194 in total.
process P5  runs 519.605 in total.
process P6  runs 461.394 in total.
process P7  runs 539.532 in total.
process P8  runs 467.869 in total.
process P9  runs 505.951 in total.
```

2. FIFO_1 的部分，誤差在+10%內

```
s@s-VirtualBox:~/OS_Project1$ cat OS_PJ1_Test/FIFO_1.txt
FIFO
5
P1 0 500
P2 0 500
P3 0 500
P4 0 500
P5 0 500s@s-VirtualBox:~/OS_Project1$ python calculate.py FIFO_1
a time unit is 0.00195645251274 sec.
process P1  runs 515.438 in total.
process P2  runs 523.130 in total.
process P3  runs 528.539 in total.
process P4  runs 548.237 in total.
process P5  runs 543.800 in total.
```

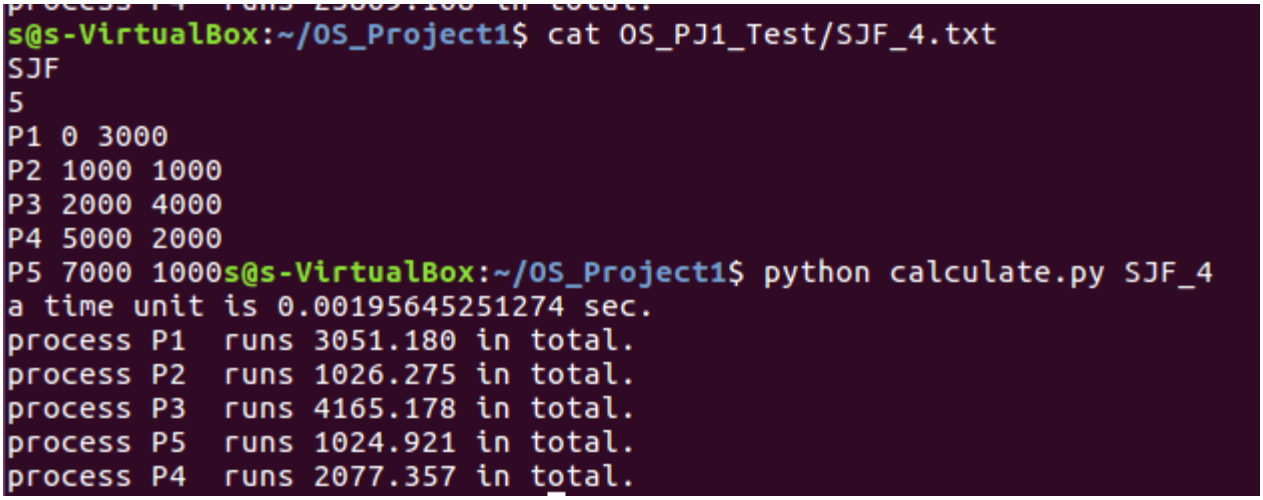
3. PSJF_2 的部分，因為有 preempt，因此 P1 的總執行時間為 4000，P2 為 1000，P3 為 7000，P4 為 2000，P5 為 1000，可看到誤差皆在 5%內

```
s@s-VirtualBox:~/OS_Project1$ cat OS_PJ1_Test/PSJF_2.txt
PSJF
5
P1 0 3000
P2 1000 1000
P3 2000 4000
P4 5000 2000
P5 7000 1000s@s-VirtualBox:~/OS_Project1$ python calculate.py PSJF_2
a time unit is 0.00195645251274 sec.
process P2  runs 1039.345 in total.
process P1  runs 4099.892 in total.
process P4  runs 1956.620 in total.
process P5  runs 1050.743 in total.
process P3  runs 7119.949 in total.
```

4. 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

```
s@s-VirtualBox:~/OS_Project1$ cat OS_PJ1_Test/RR_3.txt
RR
6
P1 1200 5000
P2 2400 4000
P3 3600 3000
P4 4800 7000
P5 5200 6000
P6 5800 5000
s@s-VirtualBox:~/OS_Project1$ python calculate.py RR_3
a time unit is 0.00195645251274 sec.
process P3  runs 14395.989 in total.
process P1  runs 19011.580 in total.
process P2  runs 18001.330 in total.
process P6  runs 20672.610 in total.
process P5  runs 24316.237 in total.
process P4  runs 25869.168 in total.
```

5. 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。