

# OS project1 report

---

資工三 B06902066 蔡秉辰

## 設計

事先說明：在本次作業中，我的 **child process** 的 **start time** 會是其被產生的時間。

## 程式運行的位置

這次作業中，為希望 scheduler (parent process) 和創建出來的其他程式 (child process) 互相影響，parent process 和 child process 會分別放在兩顆不同的 CPU 運行，兩個分別使用 0 號和 1 號 CPU。

## 程式流程介紹

在主程式讀完輸入的資訊後，我會將其依據其 ready time 做排序。接著會開始執行 scheduler 需做的事情，並將自己的 CPU 設置成 0 號。接著 scheduler 會進入 while 迴圈，每一個迴圈即代表一個 unitime。每次迴圈中都會檢查是否有程式已經準備好了(即 ready time 為現在的時間)，若準備好的話我就會 fork 一個 process，並馬上將其停止運行。接著若現在為 preemptive 或是已沒有 child process 正在執行，我將會依據 policy 來選擇下一個應該要跑的程式為何，並叫它可以開始運行。之後我會使 parent process 跑一個 unitime 迴圈的，最後更新一些資訊(e.g. 現在正在跑的 process 還剩多久會跑完)。若 policy 為 preemptive shortest job first 或是 policy 為 round-robin 且已到達 time quantum，parent process 會將其停止運行。結束以上後，就會進到下一個新的迴圈。順帶一提，一個迴圈所需時間將會是比一個 unitime 多上一點點。在最後所有 child process 均跑完後，我會呼叫程式數量的 wait() 來處理 zombie process，並結束程式。

## child process 相關

首先，上述有提到 parent process 會控制 child process 開始/停止運行，其使用的方式為使用 sched\_setscheduler 去調整其 priority。若是要開始運行則會是將 priority 設成 99，反之則調成 1。接著說明一下 child process 做的事情，在 child process 被 fork 出來之前，就會先呼叫 system call 來取得現在的時間後，才會執行 fork (此時間即為創建的時間)。fork 後，child process 做的事情為跑完一個 unitime 的迴圈，呼叫 system call 來取得結束的時間，最後印到 dmesg 內；而 parent process 做的事情則是在 stdout 印出 child process 的名字和 PID。

## 核心版本

- Ubuntu 16.04
- linux version == 4.14.25

## 與理論結果之比較

### 理論時間的計算方式

我首先使用 `TIME_MEASUREMENT.txt` 來得出平均每個 unitime 的時間。將實際運行結果的時間的最小值去對應 input ready time 的最小值。再去算出剩下的理論可開始執行之時間和理論結束時間。

### 造成差異的原因

我將四種 policy 各取一個來做比較(同 demo 的 tasks)，詳細結果在下面。主要可以觀察出以下兩個情形

1. 實際產生的時間幾乎都略晚於理論時間：這是因為 scheduler 的 while 迴圈實際上會需要略大於 1 個 unitime 所造成的。
2. PSJF 的誤差比例較大：我認為這是因為在實作上，我每個迴圈都會重新找一次現在應該要讓哪個 child process 執行並讓它運行，且在每個 scheduler 迴圈的最後都會將正在執行的 child process 擋住。這會使誤差較其他三個方式來得大。(註：RR\_3 的結果看起來誤差也不小，但我認為其中有一部份原因是因其所有程式總執行時間要花較多的 unitime，導致誤差會累加的更多)

其它可能造成誤差的因素：

1. 每一次跑一個 unitime 其實都不一樣，用 TIME\_MUSUREMENT.txt 測出來的也只是個平均。故用平均去計算的理論時間就算是在不考慮其他任何條件下，仍然不會和實際時間相同。
2. 上述有提到，實作 block child process 是藉由將 priority 調成 1。假設以下測資：

```
SJF
2
P1 0 1000
P2 0 100
```

理論上在執行 P2 前 P1 不能夠運行，但因創建 P1 時，CPU 1 並沒有其他程式，故即使其 priority 極低，因沒有其他程式在 CPU 1 搶資源，P1 還是會執行直到 scheduler 將 P2 的 priority 調高。這種偷跑的情形亦會稍微造成誤差。

3. 在 preemptive 的情形下，scheduler 會需要多次的要求 child process 停止/開始運行。但因為其一個迴圈時間是略大於一個 unitime，故它去調整其運行與否的時間會有一點誤差。

## 詳細理論 & 實際數據

### FIFO\_1

理論值為：

```
P1 1587829095.1360457 1587829096.1531935
P2 1587829095.1360457 1587829097.1703415
P3 1587829095.1360457 1587829098.1874893
P4 1587829095.1360457 1587829099.204637
P5 1587829095.1360457 1587829100.2217848
```

實際值為：

```
P1 1587829095.136045664 1587829096.151744801
P2 1587829095.136309520 1587829097.169764366
P3 1587829095.136447112 1587829098.175357056
P4 1587829095.136564942 1587829099.176771167
P5 1587829095.136682134 1587829100.198515627
```

### PSJF\_2

理論值為：

```
P2 1587829432.635011 1587829434.6693065
P1 1587829430.6007152 1587829438.7378979
P4 1587829440.7721937 1587829444.840785
```

P5 1587829444.840785 1587829446.8750808

P3 1587829434.6693065 1587829452.977968

實際值為：

P2 1587829432.811241301 1587829434.797475994

P1 1587829430.600715193 1587829438.570751682

P4 1587829441.179493016 1587829445.131503837

P5 1587829445.163472215 1587829447.160012495

P3 1587829434.894891863 1587829452.477456248

### RR\_3

理論值為：

P3 1587829553.3703225 1587829583.0710397

P1 1587829548.4880128 1587829587.139631

P2 1587829550.9291675 1587829588.1567788

P6 1587829557.845773 1587829603.4139967

P5 1587829556.6251955 1587829607.482588

P4 1587829555.8114772 1587829609.5168839

實際值為：

P3 1587829553.689071746 1587829584.129208164

P1 1587829548.488012870 1587829587.982403367

P2 1587829551.095643539 1587829588.978992282

P6 1587829558.440481964 1587829604.230523176

P5 1587829557.171807165 1587829608.353614715

P4 1587829556.285867830 1587829610.284822762

### SJF\_4

理論值為：

P1 1587829832.3135684 1587829838.4164555

P2 1587829834.3478642 1587829840.450751

P3 1587829836.3821597 1587829848.587934

P5 1587829846.5536382 1587829850.6222296

P4 1587829842.4850469 1587829854.6908212

實際值為：

P1 1587829832.313568368 1587829838.540647165

P2 1587829834.448758310 1587829840.607845990

P3 1587829836.596322875 1587829848.912323917

P5 1587829847.239622381 1587829851.412674318

P4 1587829843.085754455 1587829855.241452603