# OS_Project 1

b06705059 資工三 魏任擇

## 設計

### **FIFO**：

對每個先 arrive 的 process 都排隊進 CPU 的 ready queue 裡，按照抵達的順序執行，也就是先來的先執行。

方式：

1. 先對所有讀進來的 process 進行 sort，按照抵達的時間(R)由小到大
2. 於 main process 裡計時，一旦發現現在有 process 到達了(main_clock == R)，那就 fork()一個 child 並將它綁在 CPU0 上。
3. 將正在跑的那個 process priority 設最大(max)，其餘 priority 一律都為一樣(max - 1)。
4. 而所有時間都是在 main 裡計時，一但發現有人應該要結束了(剩餘執行時間已經到 0 了)，就使 child 結束並計算時間輸出到 kernel 上。
5. 如此一來使用 sched_setscheduler 就會執行 FIFO 的順序。

### **RR**：

每隔一定的時間就要換人跑 CPU，以達到公平的原則，其餘與 FIFO 無異。

方式：

1. 對所有讀進來的 process 進行 sort，按照抵達的時間(R)由小到大
2. 於 main process 裡計時，一旦發現現在有 process 到達了(main_clock == R)，那就 fork()一個 child 並將它綁在 CPU0 上。
3. 將正在跑的那個 process priority 設最大(max)，其餘 priority 一律都為一樣(max - 1)。
4. 多紀錄一個 cur_process_clock 變數，負責記錄目前在跑的這隻程式跑了多久，一旦發現已經跑了 time quantum = 500 unit time，這支程式還沒結束的話，就將要跑的下一個程式的 priority 調到 max，再把目前這支程式的 priority 調到 max-1，如此一來 CPU0 就會自動換人做。
5. 一旦發現有人應該要結束了(剩餘執行時間已經到 0 了)，就使 child 結束並計算時間輸出到 kernel 上。

**SJF**：

每當一個程式結束後，CPU 會先選一個最小執行時間的程式來跑(從目前已抵達的程式 ready queue 裡挑一個)，跑到這程式結束在挑下一個最小的。

方式：

1. 對所有讀進來的 process 進行 sort，按照抵達的時間(R)由小到大，若 R 相同，按照 T 的大小排序，一樣由小到大。
2. 於 main process 裡計時，一旦發現現在有 process 到達了(main_clock == R)，那就 fork()一個 child 並將它綁在 CPU0 上。
3. 將正在跑的那個 process priority 設最大(max)，其餘 priority 一律都為一樣(max - 1)。
4. 一旦發現有人應該要結束了(剩餘執行時間已經到 0 了)，就使 child 結束並計算時間輸出到 kernel 上。
5. 在讓 child 結束之前，會去看 CPU0 上還有哪些 process，找到最小的那個 process，找到並將其權限提高，才能讓 child 結束。
6. 如此一來，一旦 child 結束後，CPU0 就會自己去挑到最小的那個 process 並執行它。

**PSJF**：

每次有個程式抵達或是結束時，都會去找目前 ready queue 裡最小的那個 process，並 context switch 到那個 process 執行。

方法：

1. 對所有讀進來的 process 進行 sort，按照抵達的時間(R)由小到大，若 R 相同，按照 T 的大小排序，一樣由小到大。
2. 於 main process 裡計時，一旦發現現在有 process 到達了(main_clock == R)，那就 fork()一個 child 並將它綁在 CPU0 上，並且去 ready queue 裡找到最小剩餘執行時間的 process，將它的 priority 提高到 max，再把原本在執行的程式 priority 調低到 max-1
3. 將正在跑的那個 process priority 設最大(max)，其餘 priority 一律都為一樣(max - 1)。
4. 一旦發現有人應該要結束了(剩餘執行時間已經到 0 了)，就使 child 結束並計算時間輸出到 kernel 上。
5. 在讓 child 結束之前，會去看 CPU0 上還有哪些 process，找到最小的那個 process，找到並將其權限提高，才能讓 child 結束。
6. 如此一來，一旦 child 結束後或是有人抵達時，CPU0 都會自己去挑到最小的那個 process 並執行它。

# 核心版本

環境：Linux ubuntu 16.04LTS

Kernel : linux-4.14.25

# 比較

## FIFO

測資：

```
FIFO
7
P1 0 8000
P2 200 5000
P3 300 3000
P4 400 1000
P5 500 1000
P6 500 1000
P7 600 4000
```

結果為

```
s@s-VirtualBox:~/os_project_1/output$ cat FIFO_3_*
[ 1499.500388] [Project1] 2768 1588073702.295437251 1588073716.371201061
[ 1508.124894] [Project1] 2769 1588073716.371384634 1588073724.995822007
[ 1513.323233] [Project1] 2770 1588073724.995958945 1588073730.194229006
[ 1515.061703] [Project1] 2771 1588073730.194373827 1588073731.932721648
[ 1516.843873] [Project1] 2772 1588073731.932862479 1588073733.714914365
[ 1518.584704] [Project1] 2773 1588073733.715159903 1588073735.455768825
[ 1525.534329] [Project1] 2774 1588073735.455901162 1588073742.405483674
P1 2768
P2 2769
P3 2770
P4 2771
P5 2772
P6 2773
P7 2774
```

## RR

測資：

```
RR
6
P1 1200 5000
P2 2400 4000
P3 3600 3000
P4 4800 7000
P5 5200 6000
P6 5800 5000
```

結果為

```
s@s-VirtualBox:~/os_project_1/output$ cat RR_3_*
[ 1751.920410] [Project1] 2926 1588073943.548488570 1588073968.794122023
[ 1765.809055] [Project1] 2928 1588073950.639472890 1588073982.682906140
[ 1769.278686] [Project1] 2931 1588073969.667124345 1588073986.152573744
[ 1770.126286] [Project1] 2930 1588073972.335802048 1588073987.000181790
[ 1772.751939] [Project1] 2929 1588073953.210124526 1588073989.625860731
[ 1778.793735] [Project1] 2927 1588073946.206242360 1588073995.667718012
P1 2926
P3 2928
P6 2931
P5 2930
P4 2929
P2 2927
```

## SJF

測資：

```
SJF
5
P1 0 3000
P2 1000 1000
P3 2000 4000
P4 5000 2000
P5 7000 1000
```

結果為

```
s@s-VirtualBox:~/os_project_1/output$ cat SJF_4_*
[ 1979.293553] [Project1] 3020 1588074191.134990483 1588074196.169426335
[ 1981.073640] [Project1] 3021 1588074196.169617839 1588074197.949529022
[ 1987.754738] [Project1] 3022 1588074197.949673681 1588074204.630686825
[ 1989.374202] [Project1] 3024 1588074204.630826225 1588074206.250166381
[ 1992.718080] [Project1] 3023 1588074206.250302778 1588074209.594073510
P1 3020
P2 3021
P3 3022
P5 3024
P4 3023
```

## PSJF

**測資：**

```
PSJF
5
P1 0 3000
P2 1000 1000
P3 2000 4000
P4 5000 2000
P5 7000 1000
```

結果為

```
s@s-VirtualBox:~/os_project_1/output$ cat PSJF_2_*
[ 1622.792304] [Project1] 2840 1588073837.933532101 1588073839.664632895
[ 1626.263927] [Project1] 2839 1588073836.250228167 1588073843.136295448
[ 1631.643537] [Project1] 2842 1588073844.888700940 1588073848.515962549
[ 1633.408411] [Project1] 2843 1588073848.516250293 1588073850.280855066
[ 1638.778624] [Project1] 2841 1588073843.136424273 1588073855.651132744
P2 2840
P1 2839
P4 2842
P5 2843
P3 2841
```

# Discussion

可以發現我們的排程跟理論上排程的順序是一樣的。

與理論上不同的是

1. process 跟 process 之間切換的時間點沒辦法完全吻合，我認為產生這結果的原因是因為 context switch 會產生時間上的空檔，同時我們在排程中通知 child 要換人的時候，會產生一些 instruction 時間上的浪費(main process 會處理各種資料結構的維護)，才造成這樣的差異。
2. 同樣 T(執行時間)的 process，在 CPU 的時間是不會一樣的，我認為產生這個差異的原因是因為
   (a) 每個 CPU 跑同樣的操作，也不見得會是一樣的時間。
   (b) 本機和 vm linux 上跑的不會只有我們的程式，因此如果本機上或 vm linux 上有程式要跑的時候就會跟我們搶資源，但我們計算的時間不會停，會持續累積，因此會產生差異。