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_*
[ 3962.354865] [Project1] 3024 1588085921.509671371 1588085935.600158644
[ 3971.196762] [Project1] 3025 1588085935.600329729 1588085944.442064966
[ 3976.560916] [Project1] 3026 1588085944.442186471 1588085949.806224681
[ 3978.435324] [Project1] 3027 1588085949.806363116 1588085951.680631170
[ 3980.272529] [Project1] 3028 1588085951.680768296 1588085953.517841766
[ 3982.101757] [Project1] 3029 1588085953.517998419 1588085955.347070337
[ 3989.303811] [Project1] 3030 1588085955.347244098 1588085962.549132441
P1 3024
P2 3025
P3 3026
P4 3027
P5 3028
P6 3029
P7 3030
```

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_*
[ 4219.672742] [Project1] 3165 1588088135.620238586 1588088160.449901504
[ 4222.265815] [Project1] 3163 1588088130.402824106 1588088163.042688049
[ 4223.108539] [Project1] 3164 1588088133.010928125 1588088163.885323909
[ 4237.110424] [Project1] 3168 1588088142.643014852 1588088177.886000101
[ 4240.713895] [Project1] 3167 1588088140.031843413 1588088181.489226541
[ 4242.444812] [Project1] 3166 1588088139.151693735 1588088183.220033061
P1 3163
P2 3164
P3 3165
P4 3166
P5 3167
P6 3168
```

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_*
[ 4444.553756] [Project1] 3258 1588088380.005571478 1588088385.330299470
[ 4446.307984] [Project1] 3259 1588088385.330413921 1588088387.084552644
[ 4453.400987] [Project1] 3260 1588088387.084703448 1588088394.177657394
[ 4455.193608] [Project1] 3262 1588088394.177777809 1588088395.970300425
[ 4458.631096] [Project1] 3261 1588088395.970447414 1588088399.407831032
P1 3258
P2 3259
P3 3260
P5 3262
P4 3261
```

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_*
[ 4088.179610] [Project1] 3084 1588086059.565885061 1588086061.425039490
[ 4091.907519] [Project1] 3083 1588086057.790471501 1588086065.152952935
[ 4097.413301] [Project1] 3086 1588086067.002722492 1588086070.658742061
[ 4099.172075] [Project1] 3087 1588086070.658846852 1588086072.417515205
[ 4104.542383] [Project1] 3085 1588086065.153067183 1588086077.787829504
P1 3083
P2 3084
P3 3085
P4 3086
P5 3087
```

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。