OS_Project1 Report

B05902128 鄭百凱

1. 設計

基本架構：利用單核的CPU架構來設計排程，每一個process一開始就會被assign到同一顆CPU。

以下將程式碼的設計分為schedular 和 從schedular fork出來的process 兩部分進行討論。

- schedular :
  schedular一開始的priority就會被設到最高，等到有ready process，就會fork出去，且schedular會馬上把child的priority調低，由於schedular一開始的priority就比較高，所以child不會先偷跑。

  然後每一個time unit開始前會依據scheduling policy計算這個time unit 應該要輪到哪一個process執行，並將自己的priority調低，且該process的priority調高。如果沒有ready的process，那麼schedular會自己執行一個time unit。

- process :
  child process在被assign到和schedular同一顆CPU之後就會不斷idle直到schedular認為輪到它為止。process在第一次被schedular指定使用CPU時會呼叫get_time syscall來拿到start time，然後執行一個time unit之後就把自己的priority調低，schedular的priority調高，由schedular決定再來要由誰執行。直到process把它需要執行的時間都跑完就會再度呼叫syscall來拿到end time並用另一個syscall print_sys_mesg輸出到dmesg。

2. 核心版本

linux 4.14.25

3. 比較實際結果與理論結果，並解釋造成差異的原因

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 500
```

TIME_MEASUREMENT_stdout.txt

```
P0 2637
P1 2638
P2 2639
P3 2640
P4 2641
P5 2642
P6 2643
P7 2644
P8 2645
P9 2646
```

TIME_MEASUREMENT_dmesg.txt

```
[  227.887587] [Project1] 2637 1588132227.196368947 1588132227.934294688
[  229.407425] [Project1] 2638 1588132228.709843120 1588132229.454168194
[  230.904984] [Project1] 2639 1588132230.221713063 1588132230.951758806
[  232.343817] [Project1] 2640 1588132231.697000435 1588132232.390609264
[  233.871275] [Project1] 2641 1588132233.160626571 1588132233.918095993
[  235.400707] [Project1] 2642 1588132234.689883828 1588132235.447563537
[  236.892494] [Project1] 2643 1588132236.205366088 1588132236.939364427
[  238.327740] [Project1] 2644 1588132237.683302485 1588132238.374651398
[  239.864310] [Project1] 2645 1588132239.136211502 1588132239.911248016
[  241.384741] [Project1] 2646 1588132240.675271314 1588132241.431706975
```

平均每500個time unit會需要0.75秒，不過像2644就花了0.79秒，2650花0.70秒，並沒有非常一致。500個time unit跑兩次可能會相差0.1秒。

FIFO_2.txt

```
FIFO
4
P1 0 80000
P2 100 5000
P3 200 1000
P4 300 1000
```

FIFO_2_stdout.txt

```
P1 2251
P2 2252
P3 2253
P4 2254
```

FIFO_2_dmesg.txt

```
[  739.408429] [Project1] 2251 1588099061.451266719 1588099181.234793202
[  746.765495] [Project1] 2252 1588099181.234997028 1588099188.591965211
[  748.068950] [Project1] 2253 1588099188.592166979 1588099189.895440718
[  749.528596] [Project1] 2254 1588099189.895664597 1588099191.355097623
```

P1要跑80000個time unit，理論上要跑80000/500 * 0.75 = 120秒，實際上跑了119.8秒。
P2要跑5000個time unit，理論上要跑5000/500 * 0.75 = 7.5秒，實際上跑了7.36秒。
P3要跑1000個time unit，理論上要跑1000/500 * 0.75 = 1.5秒，實際上跑了1.3秒。
P4要跑1000個time unit，理論上要跑1000/500 * 0.75 = 1.5秒，實際上跑了1.46秒。

這組測資的結果與理論值相去不遠。

PSJF_1.txt

```
PSJF
4
P1 0 10000
P2 1000 7000
P3 2000 5000
P4 3000 3000
```

PSJF_1_stdout.txt

```
P4 2335
P3 2334
P2 2333
P1 2332
```

PSJF_1_dmesg.txt

```
[ 1258.718588] [Project1] 2335 1588099696.502675220 1588099700.550925676
[ 1264.707897] [Project1] 2334 1588099694.969459557 1588099706.540288074
[ 1273.268149] [Project1] 2333 1588099693.617393931 1588099715.100615029
[ 1286.521485] [Project1] 2332 1588099692.110359313 1588099728.354076700
```

P1要跑25000個time unit，理論上要跑25000/500 * 0.75 = 37.5秒，實際上跑了36.24秒。
P2要跑15000個time unit，理論上要跑15000/500 * 0.75 = 22.5秒，實際上跑了21.49秒。
P3要跑8000個time unit，理論上要跑8000/500 * 0.75 = 12秒，實際上跑了11.58秒。
P4要跑3000個time unit，理論上要跑3000/500 * 0.75 = 4.5秒，實際上跑了4.05秒。

這個測資的誤差就比較大，範圍大約從3%~10%，因為PSJF會preemptive，所以如果B process 搶佔 A process，然後B process又有誤差，那這個誤差也會一起算到A process上，所以誤差就會比前面的FIFO大。

觀察

有preemptive的policy誤差會比較大，因為一個誤差可能會導致多個process的誤差。例如RR跟PSJF。

由FIFO的測資中可以觀察到每次一個process結束到下一個process開始的時間大約為0.0002秒，這中間包含了context switch、schedular選擇next process的時間。

但這個誤差是每一個time unit都會有的，且理論值也是用TIME_MEASUREMENT.txt來計算的，所以推測這不是誤差的主要來源。CPU跑一個time unit的時間每次都不盡相同應該才是誤差的主要原因，這可能跟當時的CPU使用率有關，或許同時CPU剛好有其他事情要處理，例如滑鼠的I/O等等。