OS Project 1 - Process Scheduling

b07902004 資工二 陳品臻

一、設計

程式架構

main

- 讀入資料，並且將process按照到達時間排序後，呼叫scheduler

scheduler

- 永遠在其中一顆CPU上運行，負責排程
- 每單位時間都重複以下步驟：
  - 檢查上個單位時間執行的process是否已執行完成
  - 檢查是否有process在這個時間點到達，若有，就創立這個process
  - 根據指定的排程演算法，找出這個單位時間要執行的程式，並由sched_setscheduler調高要執行的process的priority
- 特別注意，所有process的priority都是由scheduler進行調整，不會發生process自己呼叫sched_setscheduler的情形

process

- 由scheduler創立出來的子程式
- 執行完指定的時間後，在dmesg印出資訊並結束

演算法

FIFO

- 原理：先到先執行
- 實作：因為一開始已經按照到達順序排序，因此每單位時間都只需要檢查「目前的process是否做完」和「下個程式是否已創立」，時間複雜度為O(1)

SJF

- 原理：剩餘最短的先執行，且不可搶佔(non-preemptive)
- 實作：若目前的process尚未完成，時間複雜度為O(1)，若已完成，要線性尋找下個程式，時間複雜度為O(n)，整體來說尋找下個程式只會發生O(n)次，複雜度為O(n^2)

PSJF

- 原理：剩餘時間最短的先執行，可以搶佔
- 實作：每單位時間都必須檢查n個程式，複雜度為O(n)，可以經由heap資料結構來優化，使得每單位時間的複雜度降到O(1)，或是以「本回合是否有新的程式被創立」來控制「是否需要檢查所有程式」

RR

- 原理：限制執行週期(time quantum)，週期到了就排到queue的最尾端，換最前面的process執行
- 實作：用queue來實作，每單位時間的複雜度都是O(1)

二、核心版本

linux-4.14.25

三、範例測資執行結果比較

TIME_MEASUREMENT

- stdout

```
P0 3411
P1 3412
P2 3414
P3 3415
P4 3416
P5 3417
P6 3418
P7 3419
P8 3420
P9 3421
```

- dmesg

```
[ 7003.425267] [Project1] 3411 1588161552.107712490 1588161552.902667138
[ 7004.924685] [Project1] 3412 1588161553.654318331 1588161554.402816626
[ 7006.418838] [Project1] 3414 1588161555.142387326 1588161555.897735608
[ 7008.024591] [Project1] 3415 1588161556.713800776 1588161557.504291347
[ 7009.589240] [Project1] 3416 1588161558.292489408 1588161559.069723252
[ 7011.074976] [Project1] 3417 1588161559.800294188 1588161560.556200497
[ 7012.562001] [Project1] 3418 1588161561.289482723 1588161562.043969621
[ 7014.037036] [Project1] 3419 1588161562.765801205 1588161563.519743013
[ 7015.498059] [Project1] 3420 1588161564.237177850 1588161564.981496010
[ 7016.940718] [Project1] 3421 1588161565.687061823 1588161566.424876290
```

time unit for this machine is `0.0015226`

FIFO_1

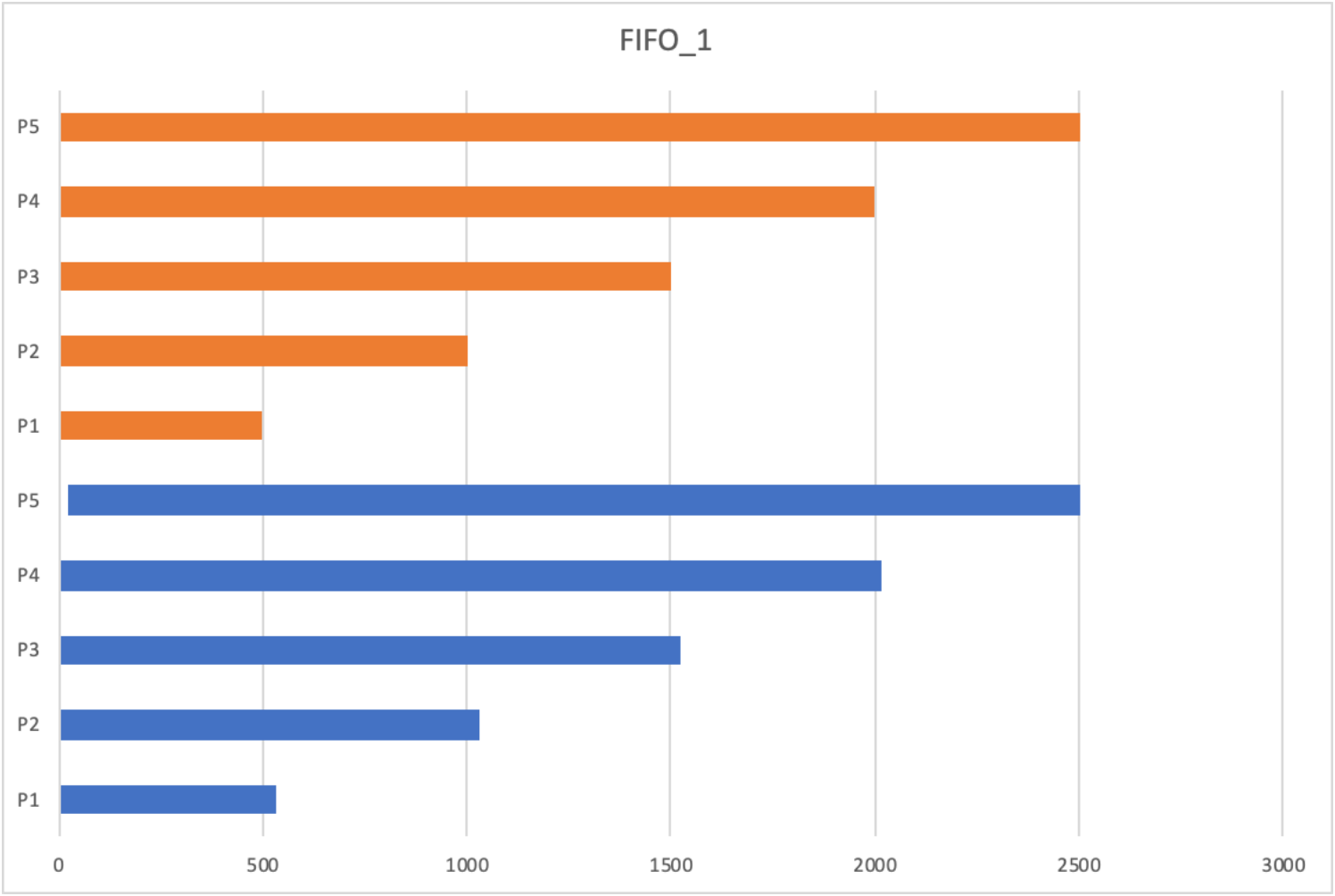
- stdout

```
P1 3440
P2 3441
P3 3442
P4 3443
P5 3444
```

- dmesg

```
[ 7260.269435] [Project1] 3440 1588161809.066699020 1588161809.875257403
[ 7261.027269] [Project1] 3441 1588161809.066842234 1588161810.633470925
[ 7261.776244] [Project1] 3442 1588161809.066801634 1588161811.382820409
[ 7262.529728] [Project1] 3443 1588161809.078277644 1588161812.136680892
[ 7263.267206] [Project1] 3444 1588161809.104841186 1588161812.874528019
```

- 比較：橘色為理論值，藍色為實際結果 error = 22.78273573 time unit



PSJF_2

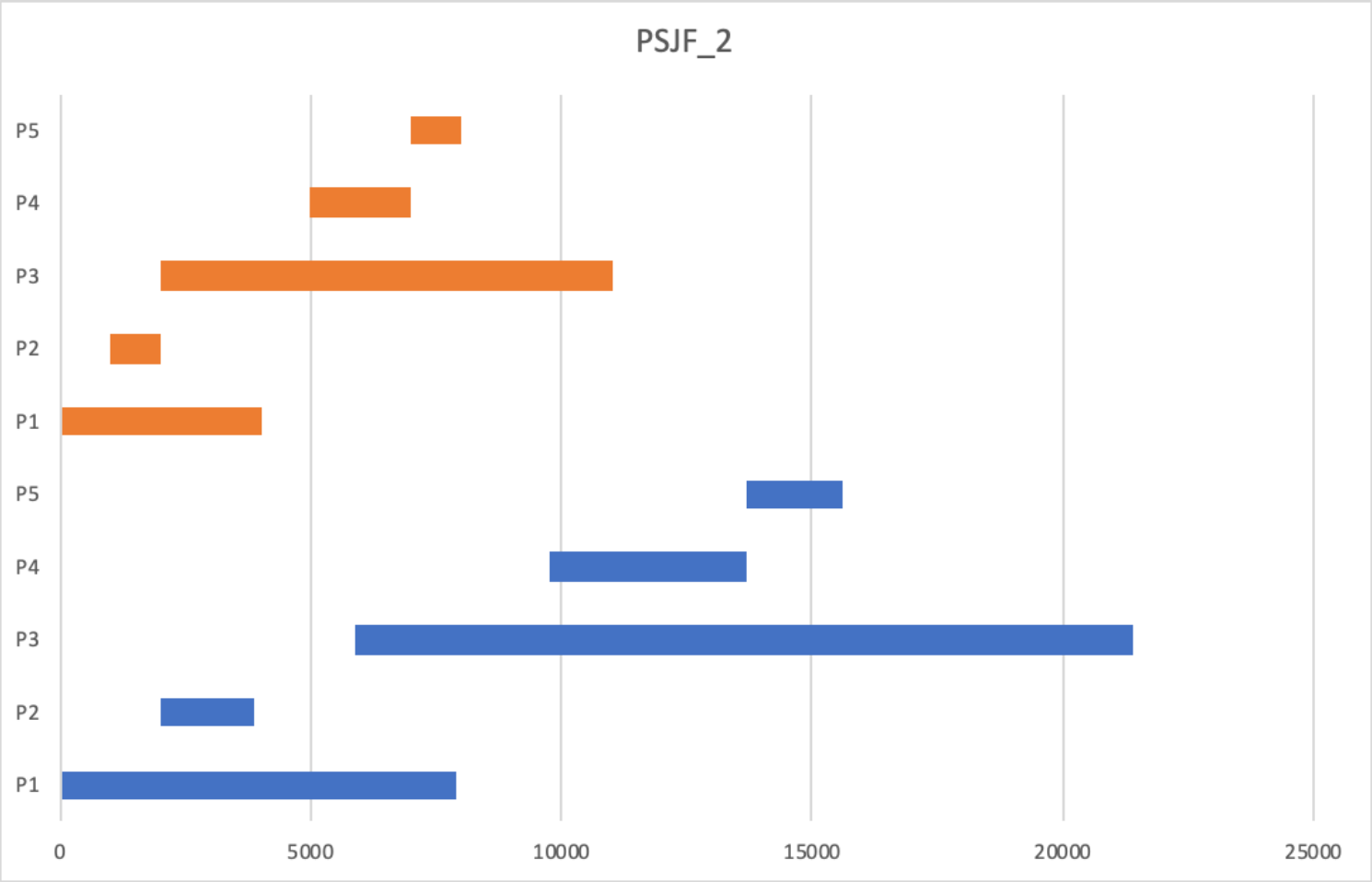
- stdout

```
P2 3674
P1 3673
P4 3678
P5 3679
P3 3675
```

- dmesg

```
[ 7988.703862] [Project1] 3674 1588162535.824813290 1588162538.673899960
[ 7994.822401] [Project1] 3673 1588162532.786343947 1588162544.795497546
[ 8003.624698] [Project1] 3678 1588162547.657585061 1588162553.602195965
[ 8006.597197] [Project1] 3679 1588162553.613462992 1588162556.576180804
[ 8015.408689] [Project1] 3675 1588162541.737250336 1588162565.392078210
```

- 比較：橘色為理論值，藍色為實際結果 error = 2828.876927 time unit



RR_3

- stdout

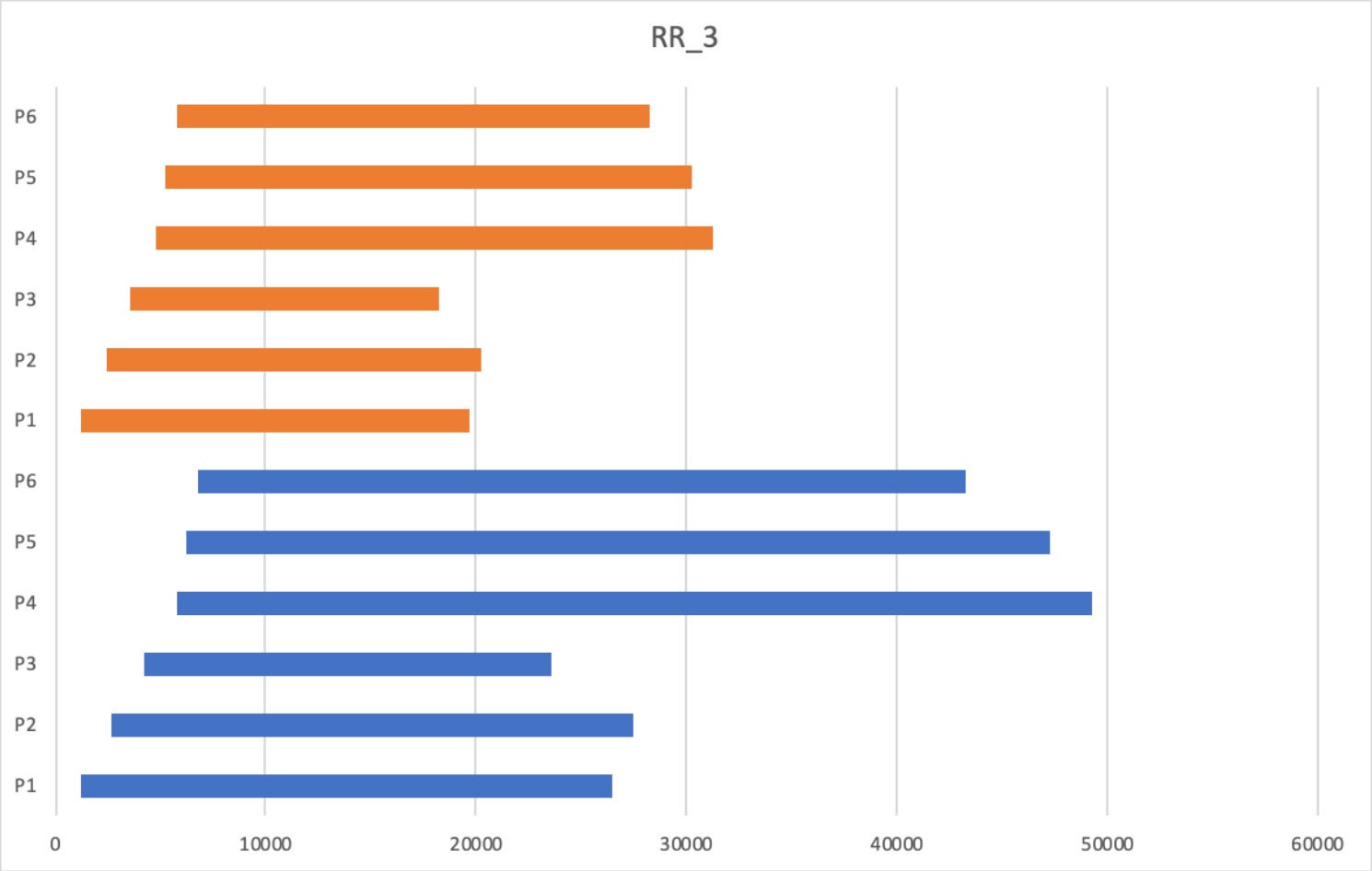
```
P3 3539
P1 3537
P2 3538
P6 3542
P5 3541
P4 3540
```

- dmesg

```
[ 7571.195049] [Project1] 3539 1588162091.509057765 1588162120.956334317
[ 7575.562190] [Project1] 3537 1588162086.946550522 1588162125.325656518
[ 7577.091446] [Project1] 3538 1588162089.225051452 1588162126.855677749
[ 7601.187079] [Project1] 3542 1588162095.364897195 1588162150.963357498
```

```
[ 7607.373617] [Project1] 3541 1588162094.600860518 1588162157.152989891
[ 7610.349128] [Project1] 3540 1588162093.992881593 1588162160.129987688
```

- 比較：橘色為理論值，藍色為實際結果 error = 10932.70571 time unit



SJF_4

- stdout

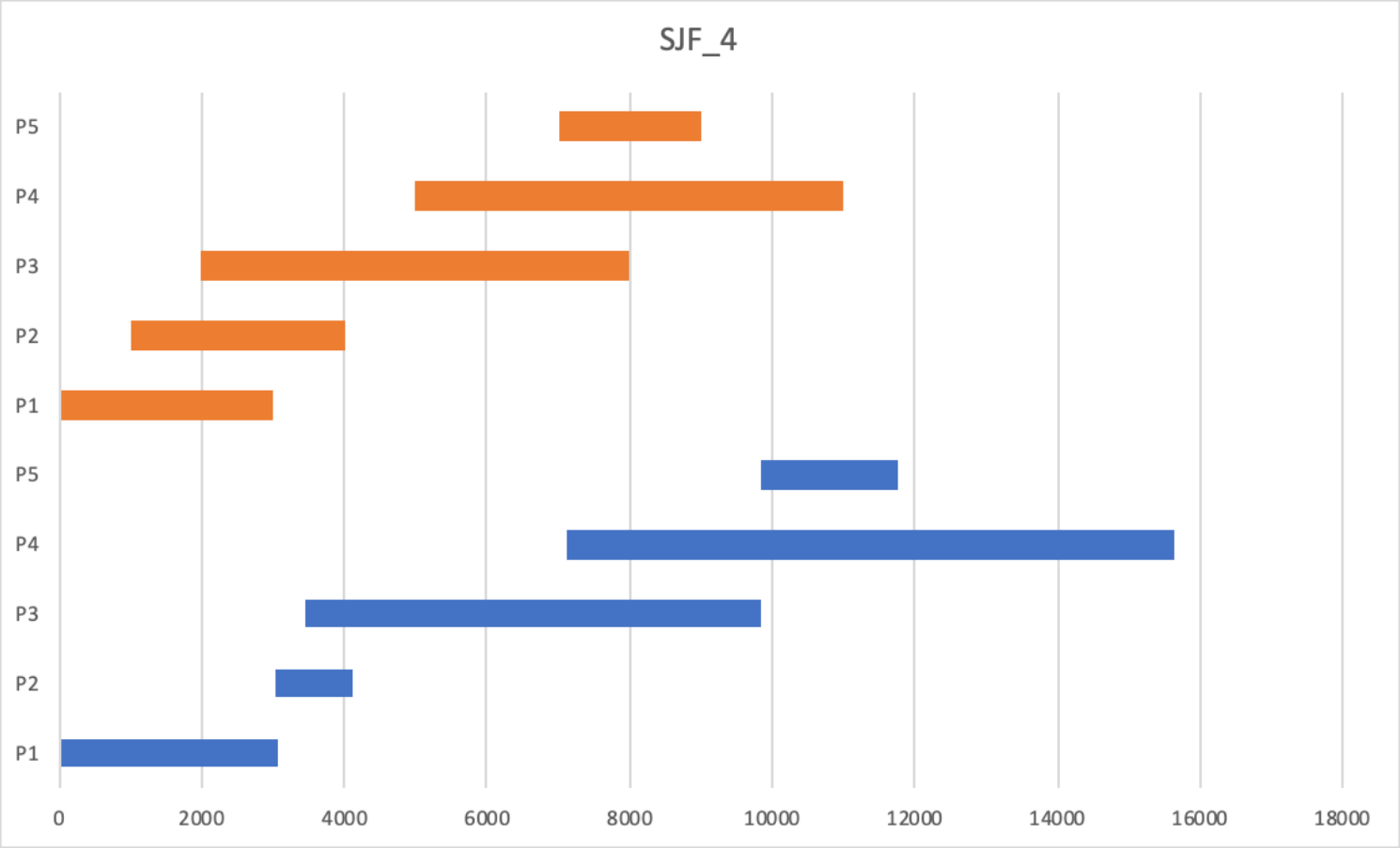
```
P1 3634
P2 3635
P3 3636
P5 3638
P4 3637
```

- dmesg

```
[ 7879.563390] [Project1] 3634 1588162424.800100465 1588162429.478858904
[ 7881.136149] [Project1] 3635 1588162429.418143751 1588162431.052404177
```

```
[ 7889.874971] [Project1] 3636 1588162430.072768564 1588162439.795593276
[ 7892.789117] [Project1] 3638 1588162439.795893534 1588162442.711196503
[ 7898.685687] [Project1] 3637 1588162435.653195323 1588162448.610714341
```

- 比較：橘色為理論值，藍色為實際結果 error = 996.1340314 time unit



四、結論

差異原因

1. scheduler的程式比較複雜，包含了調整priority、檢查有哪些程式要被創造、哪些該結束等的時間等
2. scheduler和process兩者之間沒有同步，且因為在不同的CPU執行，TIME_UNIT可能會有差異，導致兩者的時間有誤差
3. scheduler在執行context switch時，仍然會計算時間
4. 電腦上可能有其他程式在執行，造成context switch，然而執行的時間也會被算進去
5. 不同的排程演算法實作上複雜度不同，也會影響誤差