OS Project1 Report

B05902074 資工四 魏佑珊

I. 程式設計

main.c 負責讀入 scheduling type (FIFO、RR、SJF、PSJF)，並 execl 對應的程式(FIFO.c、RR.c、SJF.c、PSJF.c)。以上四程式的共用函式定義在 utils.h 與 utils.c；子程序則實現在 process.c。另外尚有 heap 以及 queue 的實作在對應的程式檔。以下分 FIFO、SJF、RR、PSJF 分開敘述如何實作。四隻程式的共同點在於都會先將輸入的 process 資訊依照 ready time sort 過，並在對應的時間點 fork 出子程序。

A. FIFO
   1. First in, First out。程式執行的順序就是一開始根據 ready time sort 好的順序。沒有其他排序方式需要執行。

B. SJF
   1. Shortest Job First。最短的程序優先，但只要有程序在執行，便不可被打斷。
   2. 我是用 min heap 來實作比較 process 的長度。每次有新的 process ready 了，除了 fork 之外，還會把 process 資訊也插入 min heap。因此，每次要決定下一個優先執行的 process 是哪一個時，只要從 heap 頂端拿就可以了。
   3. 另外，我會把預計待執行的 process 設成 PRIORITY_MIDDLE。這樣如果正在執行的小孩執行完畢，但 scheduler 還來不及 schedule 下一個該跑的程序時，被設成 PRIORITY_MIDDLE 的子程序會馬上接著執行，而不是隨意挑選一個子程序執行，順序也就不會錯。會加上這個 feature，主要是在 SJF 中，scheduler 和 child 的時間差似乎更明顯了，需要用更多機制來維護子程序的順序。
   4. 同樣也是為了解決 scheduler 和 child 時間有時會不同步的問題，我會在 child 要結束時發送 signal 給 scheduler，讓 scheduler 及時檢查是否有應該 ready 的 process 還沒被插入 heap 中。這是為了解決 P1 結束的時間點等於 P2 ready 的時間點，且 P2 又是下一個應該執行的 process 時，很容易有時間誤差而錯過。

C. PSJF
   1. Preemptive shortest job first。大致上與 SJF 的結構類似。Preemptive 的部分，我是在每次有 process ready 時，都在 heap 確認一下新加入的 process 執行時間是否比正在執行的 process 剩餘的時間短，如果是，就把正在執行的 process 替換下來。

D. RR
   1. Round-robin。每個 process 要分到 500 個 unit 的時間。在 RR 中我用

了 queue 來記錄 process 執行的輪替順序。在每次 time unit 中都去確認當前 process 是否已執行了 500 的 unit time，若是，將之移到 queue 的末端，並把執行權給 queue 最前端的 process。同時，每次都會把下一個待執行的 process 優先權先升至 PRIORITY_MIDDLE，因此若有 process 還沒執行到 500 個 unit time 就結束了，有次高優先序的 process 會自然接上去執行，順序才不會錯。

II. 核心版本

就是助教在 HW1 中提供的 4.14.25

III. 比較與討論

以下以 demo 時所用的四個檔案做比較，分別印出 total error (sec)(總執行時間誤差)、average error(sec)、和 error rate (總誤差秒數 / 總預期執行時間)

FIFO_1.txt:

```
process 1 have 0.15520033836364744 seconds delay
process 2 have 0.13007221221923826 seconds delay
process 3 have 0.12869915962219225 seconds delay
process 4 have 0.08655462265014657 seconds delay
process 5 have 0.13108024597167978 seconds delay
total_error: 0.6316065788269043
average error: 0.12632131576538086
error rate (total error / total expected execute time): 18.68593209554213%
```

PSJF_2.txt:

```
process 1 have 1.4922755241394041 seconds delay
process 2 have 0.3280598640441894 seconds delay
process 3 have 2.657846117019652 seconds delay
process 4 have 1.0085901260375962 seconds delay
process 5 have 0.3620678901672374 seconds delay
total_error: 5.848839521408079
average error: 1.1697679042816158
error rate (total error / total expected execute time): 28.839423621832996%
```

RR_3.txt:

```
process 1 have 4.324143838882446 seconds delay
process 2 have 3.8018727302551234 seconds delay
process 3 have 3.8703134536743136 seconds delay
process 4 have 6.832931995391846 seconds delay
process 5 have 5.4968346118926945 seconds delay
process 6 have 4.476138114929199 seconds delay
total_error: 28.802234745025622
average error: 4.80037245750427
error rate (total error / total expected execute time): 18.053121638169088%
```

SJF_4.txt:

```
process 1 have 0.5588203430175778 seconds delay
process 2 have 0.2672936439514162 seconds delay
process 3 have 0.9312313079833983 seconds delay
process 4 have 0.6819361686706547 seconds delay
process 5 have 0.2257117748260491 seconds delay
total_error: 2.664993238449096
average error: 0.5329986476898192
error rate (total error / total expected execute time): 17.918908685368%
```

我認為會造成理論值和實際值誤差的因素有以下幾點:

A. Scheduler 要做的事情多，例如檢查是否有新的 process ready 了、調整資

料結構等等。會造成 scheduler 和子程序的時間誤差，而要同步兩者使得子程序的執行順序不出錯，會造成 scheduler 的額外工作負擔，看起來就像是 context switch 的時間變長了。

B. 系統本身 context switch 的時間。子程序之間交替的時間本身就是 overhead。

C. 上述四個測資中，PSJF 的誤差值最大，我想是因為這筆測資剛好使得 A 及 B 提及的情況比較嚴重。這筆測資中，P4、P5 剛好都是在另一個 process finish 的時候 ready，就涉及同步問題；另外，PSJF 也會造成比 FIFO 和 SJF 更多的 context switch overload。