第二次作業

排程

1. 開發環境

Window 10

使用visual Studio Code 環境

程式語言Python

1. 實作方法及流程

相同部分:

1.皆使用一個二維變數datas存入讀入檔案，並依照一第條件arrival time, 二第條件PID排序

2.Turnaround time 計算: 結束時間(從甘特圖找)-抵達時間

3.Waiting time 計算: Turnaround time – CPU Burst

* 1. FCFS

每個時間點都先看有沒有process要進來做事，若有則先將process排到queue，當CPU有Process正在執行，則將Process ID 存到[甘特圖]，當CPU的Process做完、為空，且Queue內有Process在等待則從Queue抓第一個Process出來進CPU，若無則將`-`存到[甘特圖]，如此反覆直到CPU做完、que為空且再也沒有Process要進來，結束

* 1. RR

每個時間點都先看有沒有process要進來做事，若有則先將process排到queue，當CPU有Process正在執行，則將Process ID 存到[甘特圖]，當CPU的Process做完或Time Up，且Queue內有Process在等待則從Queue抓第一個Process出來進CPU，若無則將`-`存到[甘特圖]，如此反覆直到CPU做完、que為空且再也沒有Process要進來，結束

* 1. SJF

每個時間點都先看有沒有process要進來做事，若有則先將process排到queue並依CPU Burst 排序，當CPU有Process正在執行，則將Process ID 存到[甘特圖]，當CPU的Process做完、為空，且Queue內有Process在等待則從Queue抓第一個Process出來進CPU，若無則將`-`存到[甘特圖]，如此反覆直到CPU做完、que為空且再也沒有Process要進來，結束

* 1. SRTF

每個時間點都先看有沒有process要進來做事，若有則先將process排到queue並依CPU Burst 排序，當CPU有Process正在執行，且CPU中的Process 小於 Queue的第一個Process則將Process ID 存到[甘特圖]，否則Queue的第一個Process奪取進入CPU，被替換的Process回Queue排隊(依CPU Burst 排序)，當CPU的Process做完、為空，且Queue內有Process在等待則從Queue抓第一個Process出來進CPU，若無則將`-`存到[甘特圖]，如此反覆直到CPU做完、que為空且再也沒有Process要進來，結束

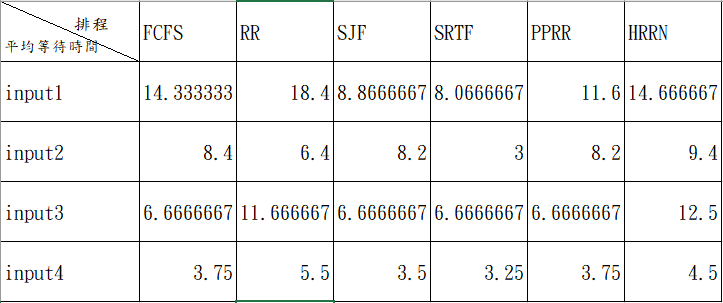
* 1. PPRR

每個時間點都先看有沒有process要進來做事，若有則先將process排到queue並依Priority 排序，當CPU有Process正在執行，且CPU中的Process 小於 Queue的第一個Process則將Process ID 存到[甘特圖]，否則Queue的第一個Process奪取進入CPU，被替換的Process回Queue排隊(依Priority 排序)，當CPU中的Process 等於 Queue的第一個Process 則使用time Slice交互使用CPU，當CPU的Process做完、為空，且Queue內有Process在等待則從Queue抓第一個Process出來進CPU，若無則將`-`存到[甘特圖]，如此反覆直到CPU做完、que為空且再也沒有Process要進來，結束

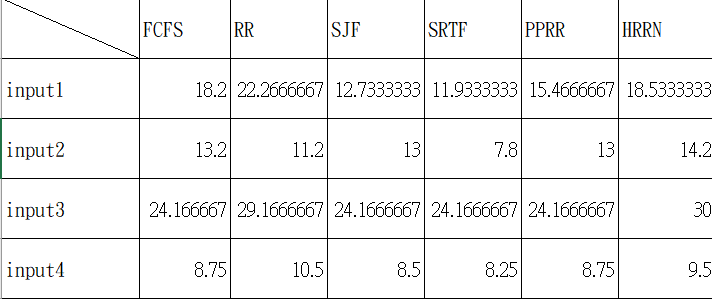
* 1. HRRN

每個時間點都先看有沒有process要進來做事，若有則先將process排到queue並依[highest Response Ratio]排序，當CPU有Process正在執行，則將Process ID 存到[甘特圖]，當CPU的Process做完、為空，且Queue內有Process在等待則從Queue抓[highest Response Ratio]進CPU，若無則將`-`存到[甘特圖]，如此反覆直到CPU做完、que為空且再也沒有Process要進來，結束

1. 不同排程法的比較
   1. Average waiting time:



* SRTF的平均等待時間**最短**。
* Input1(Time Slice= 1)，越多Process在Queue等待，且CPU Burst都比較大時，RR的平均等待時間比較長(還沒做完就要換下一個Process做)
* 若有足夠時間提升highest Response Ratio對大CPU Burst較公平(可以比一寫小CPU Burst的 Process先進入CPU執行)，但是同樣會阻礙小CPU Burst 的Process進入的時間(等待時間增加)，所以HRRN的waiting time 相對較大。
  1. Average Turnaround time:



* 往返時間 = 抵達時間 – 結束時間 = CPU Burst + 等待時間 (沒I/O的情況下)
* 因為CPU Burst 固定，所以往返時間大小受 等待時間 影響 [正相關]
* 等待時間長，往返所需時間越大(因此作圖類似Average waiting time的作圖)
* 同一個檔案，往返時間總和是一樣的
* 一樣SRTF的average Turnaround time **最短**

1. 結果與討論:

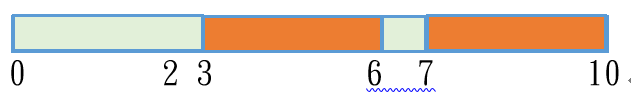
我覺得FCFS的排程法寫跟SJF、HRRN很像只差在，SJF要在Queue排序它們兩個會有各自的排序方式，再來我是先寫SRTF，因為跟SJF又有許多相似之處，指差別在可奪取的部分，再來是RR，因為是第一個需要用到time Slice的排成，所以在哪裡重新設定倒數時間，花費一段時間研究。最後PPRR是我覺得最難的部分，因為我原本跑input1、2都是對的，結果input3錯了，原因是因為我原本以為[遇到相同的Priority就要轉成RR的方式交互使用CPU]的意思是從遇到相同Priority才要開始倒數time Slice，但實際上應該是一直都要持續到數time Slice，若都沒遇到相同time Slice = 0，又再從新倒數，若遇到相同Priority，則剛好做完這個時間片段，就應該將CPU給Queue中相同Priority的那個Process做執行，才是正確的想法。

Time Slice = 3

PID Arrial CPU Burst Priority

P1 0 4 1

P2 2 6 1



還有一個我覺得很容易錯的地方，1-5的排程，Queue中除了各自排程的原本排序(CPU Burst、Priroity…)，還要記得上次要條件Arrival time,跟次次要條件PID才可以。