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

2. 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

   甲、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

   乙、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

   丙、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

   丁、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

戊、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

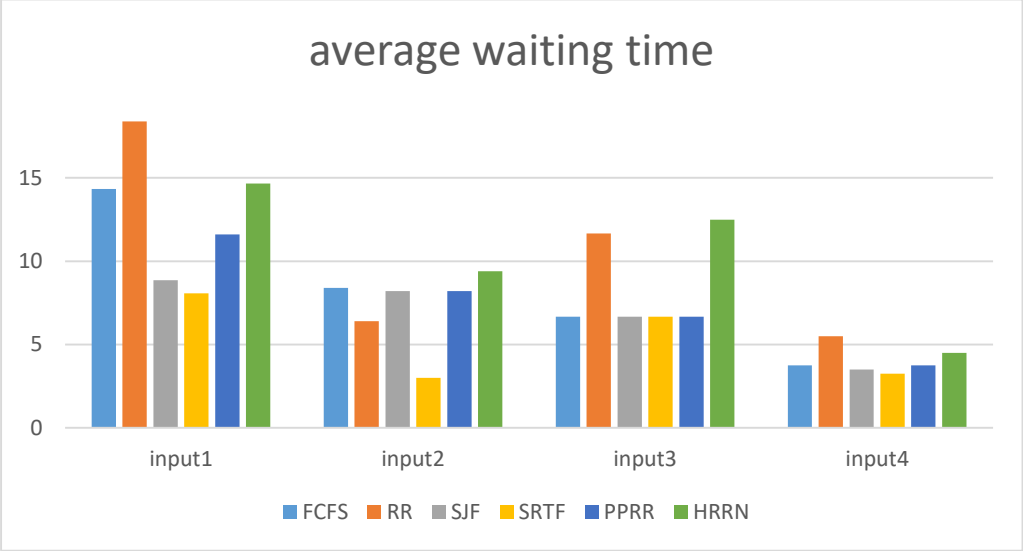
己、HRRN

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

3. 不同排程法的比較

甲、Average waiting time:

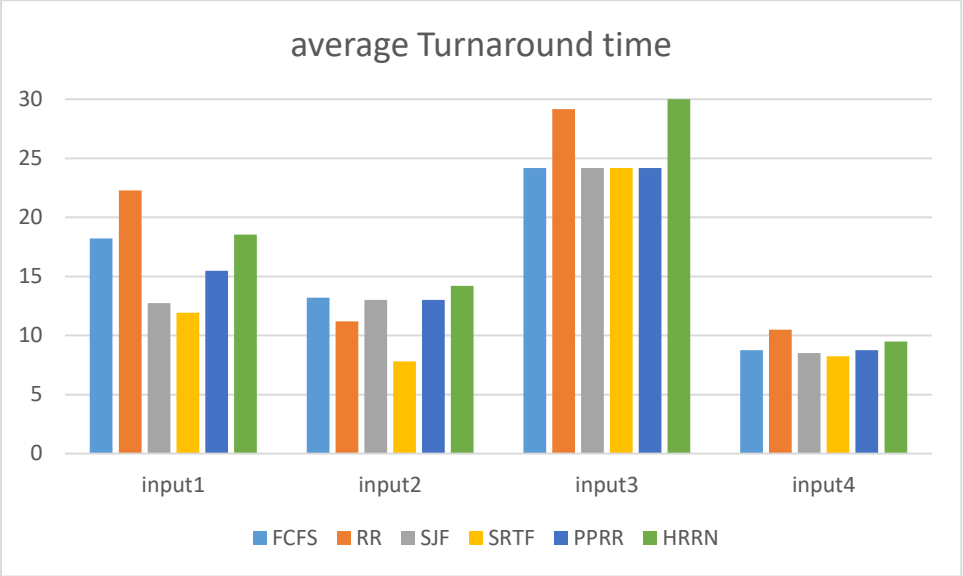
排程 / 平均等待時間	FCFS	RR	SJF	SRTF	PPRR	HRRN
input1	14.333333	18.4	8.8666667	8.0666667	11.6	14.666667
input2	8.4	6.4	8.2	3	8.2	9.4
input3	6.6666667	11.666667	6.6666667	6.6666667	6.6666667	12.5
input4	3.75	5.5	3.5	3.25	3.75	4.5



- ✧ 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。

乙、 Average Turnaround time:

	FCFS	RR	SJF	SRTF	PPRR	HRRN
input1	18.2	22.2666667	12.7333333	11.9333333	15.4666667	18.5333333
input2	13.2	11.2	13	7.8	13	14.2
input3	24.166667	29.1666667	24.1666667	24.1666667	24.1666667	30
input4	8.75	10.5	8.5	8.25	8.75	9.5



✧ 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**

4. 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。