OS Project 說明文件

資訊三甲 11020136 鄭絜馨

1. 開發環境

   Visual Studio Code(python)

2. 實作方法和流程

   - 讀檔

     process 以 class 方式儲存將所有 process 資訊存在 class 內，並用 list 存個別的 process，類似 c 裡面 struct 的概念。

   - FCFS

     先將所有 process 以 arrival time 排序，若 arrival time 相同則以 pid 較小者優先，用迴圈從第一個 process 開始往後執行，比較當前時間與現在 process 的 arrival time，若當前時間大於等於 process arrival time 就執行他，否則代表當前時間沒有 process 要使用 cpu，甘特圖就會畫上"-"，兩者都會更新當前時間，重複執行直到迴圈結束，回傳甘特圖。

   - RR

     第一個執行的 process 為 arrival time 最小的，之後以 while 迴圈每個時間為單位更新目前狀態，用 running_task(list)來存目前執行的 process 可使用 cpu 的剩餘時間，若此刻有其他 process 抵達就放入 ready queue 等待，每次迴圈會更新剩餘時間及當前時間，若 running_task 存的 process 剩餘可使用時間為 0，就會去看他 rest of cpu burst time 是否為 0，若為 0 就放入 done_process list 內紀錄，否則放回 ready queue 等待下次使用 cpu，並將 running_task 清空，並到 ready queue 尋找下一個要使用 cpu 的 process，一直重複執行直到所有 process 都被記錄到 done_process，回傳甘特圖。

   - SJF

     第一個執行的 process 為 arrival time 最小的，之後的根據 cpu burst time 排序，依序執行，若做到一半發現此刻沒有 process 執行就改以 arrival time 排序，找到下一個要做的 process 並更新 current time，重複直到所有 process 都被放到 done_process 結束，回傳甘特圖。

   - SRTF

     第一個執行的 process 為 arrival time 最小的，之後的根據剩餘 cpu burst time 排序，若相同則比較 arrival time，若又相同就再比較 pid。以 while 迴圈實作，以秒為單位重複迴圈，每次迴圈結束會重新按照當前剩餘 cpu burst time 排序，若正在使用 cpu 的剩餘 cpu

burst time 等於 0 就會加到 done_process，迴圈會在所有 process 都加到 done_process 結束，並回傳甘特圖。

- HRRN

  第一個執行的 process 為 arrival time 最小的，此 process 執行完後根據當前時間計算每個 process 的 ratio，並將計算出來的 ratio 存在一個 list，因為我有先把 process 根據 arrival time、pid 排序，所以存放的 ratio 也是有排序的，只需要從左到右取最大的，若一樣大就取最左邊的，執行該 process，每次執行完就將當前 process 放入 done_process，並清空 ratio 重新計算，重複動作直到所有 process 都加到 done_process 結束，並回傳甘特圖。

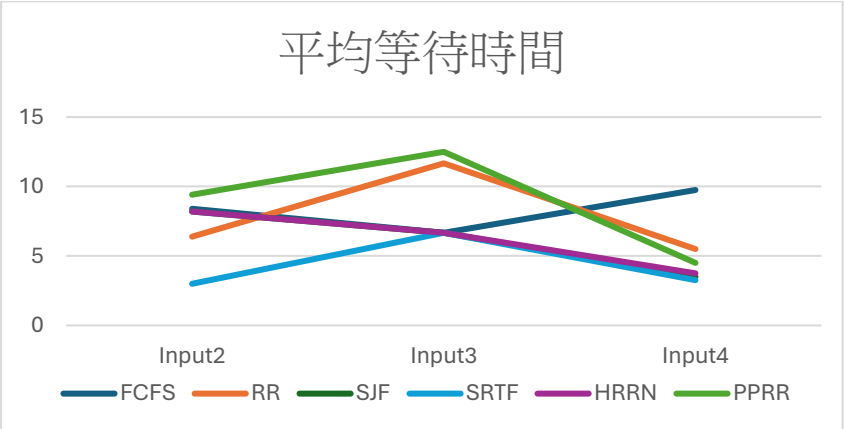
- PPRR

  第一個執行的 process 為 arrival time 最小的，使用 while 迴圈，以時間為單位，判斷當前時間有沒有 process 抵達，根據 priority 插入到 ready queue，跟 RR 不同的是進入 ready queue 的 process 不是根據抵達先後，而是先比較 priority 大小，根據 priority 插入到對應位置，因此 ready queue 是以 list 實作，同樣從 ready queue 取一個 process 到 running task 執行，running task 紀錄該 process 可使用的剩餘時間，但有可能還沒做完就被其他 process 篡位，因此每秒都會比較當前 process 的 priority 跟 ready queue 最前面的 process 大小，若 ready queue priority 優先於 running task 的 process 就會被取代，重複動作直到所有 process 都加到 done_process 結束，並回傳甘特圖。

3. 不同排程法比較

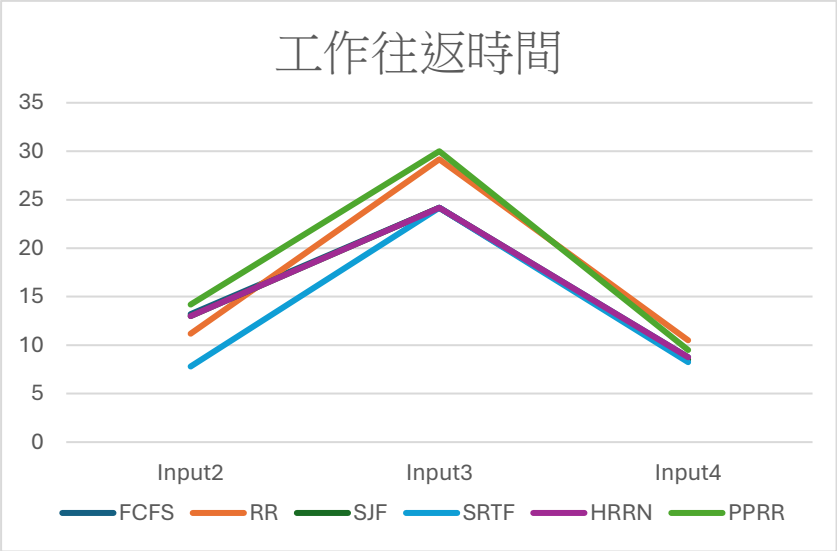
- 平均等待時間

	FCFS	RR	SJF	SRTF	HRRN	PPRR
Input2	8.4	6.4	8.2	3	8.2	9.4
Input3	6.67	11.67	6.67	6.67	6.67	12.5
Input4	9.75	5.5	3.5	3.25	3.75	4.5



- 工作往返時間

	FCFS	RR	SJF	SRTF	HRRN	PPRR
Input2	13.2	11.2	13	7.8	13	14.2
Input3	24.17	29.17	24.17	24.17	24.17	30
Input4	8.75	10.5	8.5	8.25	8.75	9.5



以下維根據三種 input 所得結論

- 效率最差: PP & PPRR

由上表可得 PP、PPRR 往返時間及平均等待時間都屬於方法裡面比較大的，主要原因可能是 process 每次只能做 time slice 所規定的時間，若時間內沒做完就要回 ready queue 等待，所以整體等待時間與往返時間會較大。而 PPRR 又比 RR 效率差的可能是新進的 process 的 priority 較小，可以搶奪先到的 process，導致等待時間又更長。

- 效率最佳: SRTF

每次執行都以剩餘時間最少的 process 優先，短時間內會完成最多 process，除非每次新進的 process cpu burst time 都很小，不然不會造成等待及往返時間過長。

4. 結果與討論

- FCFS

  Process 的 cpu burst time 越小效率越好，input3 平均 cpu burst time 較大因此效率差了一點。

- RR

  Time slice 的大小會影響 RR 的效率，要找到適合的 time slice 大小，才可以最佳化此種排程的效率，若 time slice 太大就會像 FCFS，失去 time sharing 的目的，若 time slice 太小又會造成 cpu 執行效率很差。

- SJF

  當系統中存在大量執行時間短小的 process 時，SJF 能夠最大程度地減少平均等待時間，不過卻會造成執行時間大的 process 可能會餓死。
- SRTF

  Waiting time 最小，因為它會優先執行剩餘執行時間最短的 process，而不會因為某個 process 的執行時間過長而導致其他 process 等待過久，但實作上較難實現，因為不一定可以得到 process 的 cpu 執行時間。
- HRRN

  比 SRTF 多考慮了等待時間，不會造成餓死，但每當一個 process 執行完就要重新計算一次，實務上也不一定可以得到 process 的 cpu 執行時間。
- PPRR

  RR 加上了 priority，較緊急的 process 優先執行，通常較緊急的 process 執行時間不會太長，因此可以降低平均等待時間。