程式作業二

- **開發環境**

  Visual Studio Code (使用語言： Python)

- **實作方法和流程**

  在進行排程之前，會先將 input 檔中的每個 Process 資訊(Process ID、CPU Burst、Arrival time、Priority)先依照 Process ID 由小大到排序，再依照 Arrival time 由小大到排序(這樣的用意就是如果有相同的 Arrival time，會先選擇 PID 較小的 Process)，最後將這些排序好的 Process 資訊存放在一個 global 的 dataList 中，以便後續進行各個排程時好讀取 Process 資訊。

1. FCFS

由於 FCFS 是不可被奪取且先到的 Process 可以先做，所以在此會利用迴圈的方式依序讀取在讀檔時已經依照 Arrival time 排序好的 Process 資訊。

- ✧ 會有一個變數紀錄開始時間(start_time 預設為 0)與結束時間(end_time 預設為 0)
- ➢ 當目前讀取的 Process Arrival time 小於 start_time，就是代表 CPU 目前是閒置的(在甘特圖中以"-"表示)，且會更新 start_time 為目前讀取的 Process 的 Arrival time。
- ➢ 讓目前到達的 Process 使用 CPU，且因為 FCFS 是不可被奪取，因此會直接更新此 Process 使用完 CPU 的時間(使用 end_time 紀錄，end_time = start _time + 該 Process 的 Arrival time)，得到 Process 使用完 CPU 的時間就可以進行 Turnaround Time 與 Waiting Time 的計算。
- ➢ Turnaround Time = end_time - 該 Process 的 Arrival time
- ➢ Waiting Time = Turnaround Time - 該 Process 的 CPU Burst
- ➢ 依序執行每個 Process 之後，將 Gantt chart(甘特圖)、各個 PID 與 Process Waiting Time、Process Turnaround Time 資訊寫成一個 output 檔。

2. RR

- ✧ 在此會利用迴圈的方式讀取已經依照 Arrival time 排序好的 Process 資訊。且會有一個變數紀錄開始時間(start_time 預設為 0)與結束時間(end_time 預設為 0)、ready_queue 用來記錄到達的 Process。

- ✧ 迴圈的停止條件為所有 Process 皆已到達(讀取)且 ready_queue 是空的，這表示所有 Process 皆處理完成。
  - ➢ 當目前讀取的 Process Arrival time 小於 start_time 且 ready _queue 是空的，就是代表 CPU 目前是閒置的(在甘特圖中以"-"表示)，且會更新 start_time 為目前讀取的 Process 的 Arrival time。
  - ➢ 將到達的所有 process 放入 ready_queue 中，再從 ready_queue 中取出第一個 Process，首先判斷該 Process 剩餘 CPU Burst Time 是否大於一個 time slice，如果是的話該 Process 用完一個 time slice 後會先去檢查是否有新的 Process 到達，有的話就必須讓這些到達的 Process 先排進去 ready_queue 中，然後該 Process 會排到 ready_queue 的最尾端。如果該 Process 剩餘 CPU Burst Time 小於等於一個 time slice，代表該 Process 已經處理完成，得到 Process 使用完 CPU 的時間就可以進行 Turnaround Time 與 Waiting Time 的計算。
  - ➢ 最後將 Gantt chart(甘特圖)、各個 PID 與 Process Waiting Time、Process Turnaround Time 資訊寫成一個 output 檔。

3. SJF

- ✧ 在此會利用迴圈的方式讀取已經依照 Arrival time 排序好的 Process 資訊。且會有一個變數紀錄開始時間(start_time 預設為 0)與結束時間(end_time 預設為 0)、ready_queue 用來記錄到達的 Process。
- ✧ 迴圈的停止條件為所有 Process 皆已到達(讀取)且 ready_queue 是空的，這表示所有 Process 皆處理完成。
  - ➢ 當目前讀取的 Process Arrival time 小於 start_time 且 ready _queue 是空的，就是代表 CPU 目前是閒置的(在甘特圖中以"-"表示)，且會更新 start_time 為目前讀取的 Process 的 Arrival time。
  - ➢ 將到達的所有 process 放入 ready_queue 中，對 ready_queue 中的所有 Process 依照 CPU Burst Time 由小到大排序之後，取出 ready_queue 中第一個 Process，因為 SJF 是不可被奪取，因此會直接更新此 Process 使用完 CPU 的時間，得到該時間後就可以進行 Turnaround Time 與 Waiting Time 的計算。
  - ➢ 最後將 Gantt chart(甘特圖)、各個 PID 與 Process Waiting Time、Process Turnaround Time 資訊寫成一個 output 檔。

4. SRTF

- ✧ 此方法與方法三(SJF)大致相同，唯一差別是 SJF 是不可被奪取，但 SRTF 可以被奪取的，因此每次只要有新的 Process 到達，就必須要比較所有已經到達的 Process 剩餘的 CPU Burst Time，如果有小於目前使用 CPU 的 Process 的剩餘 CPU Burst Time，則該 Process 因為被奪取，所以要將它放回到 ready_queue 中。也因為 Process 被加入到 ready_queue 中，所以 ready_queue 中的所有 Process 需要重新依照剩餘 CPU Burst Time 由小到大排序，最後直到所有 Process 皆已到達且 ready_queue 是空的，表示所有 Process 都處理完成。
- ✧ 最後將 Gantt chart(甘特圖)、各個 PID 與 Process Waiting Time、Process Turnaround Time 資訊寫成一個 output 檔。

5. HRRN

- ✧ 此方法與方法三(SJF)大致相同，唯一差別是 SJF 是依據剩餘 CPU Burst Time 由小到大排序，而 HRRN 是依據 Response Ratio 由大到小排序。
- ✧ Response Ratio =>
  ((start_time – Process Arrival Time) + Process Burst Time) / Process Burst Time
- ✧ 最後將 Gantt chart(甘特圖)、各個 PID 與 Process Waiting Time、Process Turnaround Time 資訊寫成一個 output 檔。

6. PPRR

- ✧ 此方法與方法四(SRTF)想法大致相同，唯一差別是 SRTF 是依照剩餘 CPU Burst Time 排序，而 PPRR 是依照 Priority 由小到大排序，且每次 Process 只能使用一個 time slice，也因為 PPRR 是可被奪取的，因此每次只要有新的 Process 到達，就必須要比較到達的 Process Priority，如果有小於目前使用 CPU 的 Process 的 Priority，則該 Process 因為被奪取，所以要將它放回到 ready_queue 中。因為 Process 被加入到 ready_queue 中，所以 queue 中的所有 Process 需要重新依照 Priority 由小到大排序，最後直到所有 Process 皆已到達且 ready_queue 是空的，表示所有 Process 都處理完成。
- ✧ 最後將 Gantt chart(甘特圖)、各個 PID 與 Process Waiting Time、Process Turnaround Time 資訊寫成一個 output 檔。

● 不同排程法的比較

以下表格是利用上述六個排程法對基礎測資中的 input1、input2、input3、input4 四個不同檔案中的 Process 進行排程所得之每個方法的平均等待時間(Average Waiting Time)與平均往返時間(Average Turnaround Time)，在此會透過所得結果進行不同排程法的比較。

	FCFS	RR	SJF	SRTF	HRRN	PPRR
input1	14.33	18.40	8.87	8.07	11.60	14.67
Input2	8.40	6.40	8.20	3	8.20	9.40
Input3	6.67	11.67	6.67	6.67	6.67	12.50
Input4	3.75	5.50	3.50	3.25	3.75	4.50

表 1 ： 不同 input 檔個別使用六種排程法的平均等待時間(ms)

	FCFS	RR	SJF	SRTF	HRRN	PPRR
input1	18.20	22.27	12.73	11.93	15.47	18.53
Input2	13.20	11.20	13	7.80	13	14.20
Input3	24.17	29.17	24.17	24.17	24.17	30
Input4	8.75	10.50	8.50	8.25	8.75	9.50

表 2 ： 不同 input 檔個別使用六種排程法的平均往返時間(ms)

透過上面兩個表格，可以推論出六個排程之間的差異

- FCFS :

  由於 FCFS 先抵達的 Process 會先做，所以必須等到在使用 CPU 的 Process 完成之後，下一個 Process 才可以使用 CPU，因此如果有 Process 的 CPU Burst Time 較大，後面到達且在等待的 Process 需要等很久，因此 FCFS 對於每個 Process 的 CPU Burst Time 較小時的整體表現會比較好。但藉由表 2 的 input3 六個排程結果可以得知，雖然每個 Process 的 CPU Burst Time 都蠻大的(相對 input1、2、3)，但因為每個 Process 的抵達時間差距很大，因此他在平均等待時間(Average Waiting Time)與平均往返時間(Average Turnaround Time)會相對較小。

- PPRR、RR :

  這兩個方法在這六個方法中的 Average Waiting Time 與 Average Turnaround Time 是相對較大的，因為如果在一個 time slice 之後

Process 仍須使用，需要重新到 ready queue 裡等待。因此如果 Process CPU Brust Time 較大且 time slice 較小時，Process 必須常常回到 queue 中重新排隊，所以整體 Waiting Time 與 Turnaround Time 因此會較高。

- SJF 、SRTF：

  在這六種排程中 SRTF 的 Average Waiting Time 與 Average Turnaround Time 會是最小的，因為這個方法會將最小剩餘 CPU Burst Time 先做處理，這樣一來整體的 Turnaround Time 會與 Waiting Time 會較小。

  而 SJF 會相較於 SRTF 的 Average Waiting Time 與 Average Turnaround Time 大一點的原因是因為 SJF 是不可被奪取的，因此可能會有比目前使用 CPU 的 Process 剩餘 CPU Burst Time 還小的 Process 到達，但必須要等到使用 CPU 的 Process 用完之後才可以使用，所以整體時間會多一些。

- HRRN：

  HRRN 的 Average Waiting Time 與 Average Turnaround Time 會稍微高於 SRTF 的原因是雖然它會算出所有到達的 Process Response Ratio 並由大到小排序，因此會讓等待較久的 Process 有機會可以提高他的優先順序，但是此方法不可以被奪取必須等到使用 CPU 的 Process 處理完成，因此所花費的時間會多一點。

結果與討論

透過上述的六個排程的比較，可以得知排程方式沒有最好與最壞之分，每個排程會因為 Process 的狀況(Process 的 CPU Burst、Arrival time、Priority)而導致有不同的結果(Average Waiting Time 與 Average Turnaround Time 不同)，因此在使用某個排程方法時，需要多加留意排程的規則(是否可以被奪取)與重視的地方(SJF 與 SRTF 重視 Process 的 Priority、HRRN 重視 Process 的 Response Ratio)，這些都是決定排程方法是否適合的原因!!!