OS_HW2 程式說明文件

一、開發環境

1.處理器：Intel(R) Core(TM) i7-10750H
2.記憶體：8.00 GB
3.系統類型：64 位元作業系統，x64 型處理器
4.Windows 規格：Windows 10 家用版
5.IDE：Visual Studio Code
6.使用語言：Python

二、實作方法和流程

1.讀檔：

首先請使用者輸入檔案名稱，我會先把 Method、Time Slice 分別記錄起來，接者呼叫 ReadProcess()，ReadProcess()會先讀取標題(ID, CPU Burst, Arrival Time, Priority)，之後再將 process 資訊依序儲存在 processList 中。

2.寫檔：

我寫了兩個 function(PrintResult(), PrintAllResult())，前者是給 method 1-5 使用，而後者專門處理 method 6(print ALL)，PrintResult()會先寫入 method 的名稱，再寫入該方法的甘特圖，接著依序 PID 寫入 waiting time, turnaround time，並做適當排版，PrintAllResult()其實大同小異，只是因為同時要處理 5 個 method，所以我分別使用了 5 個 class 所產生的 object，在處理時也比較方便。

3.method 1： # FCFS (First Come First Serve)

首先建立 FCFS_Process_List，並複製 processList 的內容，代表 input 資訊，接者利用 class FCFS 建立 FCFS_Simulate，之後呼叫 FCFS_Simulate.Start()進行 CPU 排程，FCFS_Simulate.Start()會先依 Arrival Time 排序所有 process，如果 Arrival Time 相同，再利用 PID 排序，確認順序後執行 CheckProcess()，CheckProcess()的目的是將抵達的 process 放進 Waiting_Queue 裡面，並將放入 Waiting_Queue 的 process 從 FCFS_Process_List 中 pop 出來，代表 process 已處裡過，接者執行 RunProcess()，RunProcess()從 Waiting_Queue 中 pop process 出來並執行，執行時將 process 的 CPU_Burst_Remaining 減一，並記錄 PID(0-9, A-Z) 於甘特圖，如果 process 的 CPU_Burst_Remaining 等於 0，代表程序執行完畢，記錄完成時間，並計算出 Turnaround_Time, Waiting_Time，接者將完成的 process 加入至

Done_List 中，完成 FCFS_Simulate.Start()後，最後呼叫 PrintResult()將結果寫檔。

4.method 2： # RR (Round Robin)
首先建立 RR_Process_List，並複製 processList 的內容，代表 input 資訊，接者利用 class RR 建立 RR_Simulate，之後呼叫 RR_Simulate.Start()進行 CPU 排程，RR_Simulate.Start()會先依 Arrival Time 排序所有 process，如果 Arrival Time 相同，再利用 PID 排序，確認順序後執行 CheckProcess()，CheckProcess()的目的是將抵達的 process 放進 Waiting_Queue 裡面，並將放入 Waiting_Queue 的 process 從 RR_Process_List 中 pop 出來，代表 process 已處裡過，在這裡要特別處裡 process 尚未完成但 timeout 的情形，因為 process 的 Time_Slice 等於 0，所以要把該 process 再重新放入 Waiting_Queue 中，接者執行 RunProcess()，RunProcess()從 Waiting_Queue 中 pop process 出來並執行，執行時將 process 的 CPU_Burst_Remaining 減 1，也把 Time_Slice 減 1，並記錄 PID(0-9, A-Z) 於甘特圖，如果 process 的 Time_Slice 等於 0，CPU_Burst_Remaining 亦為 0，代表程序執行完畢，記錄完成時間，並計算出 Turnaround_Time, Waiting_Time(Turnaround_Time 需先算出)，接者將完成的 process 加入至 Done_List 中，如果 process 的 CPU_Burst_Remaining 為 0，但 Time_Slice 不為 0，代表程序提前結束，記錄完成時間，並計算出 Turnaround_Time, Waiting_Time，接者將完成的 process 加入至 Done_List 中，完成 RR_Simulate.Start()後，最後呼叫 PrintResult()將結果寫檔。

5.method 3： # SRTF (Shortest Remaining Time First)
首先建立 SRTF_Process_List，並複製 processList 的內容，代表 input 資訊，接者利用 class SRTF 建立 SRTF_Simulate，之後呼叫 SRTF_Simulate.Start()進行 CPU 排程，SRTF_Simulate.Start()會先依 Arrival Time 排序所有 process，如果 Arrival Time 相同，再利用 CPU_Burst 排序，如果還是一樣，再利用 PID 排序，確認順序後執行 CheckProcess()，CheckProcess()的目的是將抵達的 process 放進 Waiting_Queue 裡面，並將放入 Waiting_Queue 的 process 從 SRTF_Process_List 中 pop 出來，代表 process 已處裡過，這裡有個重點要注意，因為 SRTF 是 Preemptive，所以如果有 process 剩餘的時間小於正在執行的 process 的剩餘時間，就進行搶奪，原本正在執行的 process 進入 Waiting_Queue，另外，如果剩餘時間相同，就比較 Arrival Time，如果還是相同，再比較 PID，如果成立，依然可以進行搶奪，接者

執行 RunProcess()，RunProcess()從 Waiting_Queue 中 pop process 出來並執行，執行時將 process 的 CPU_Burst_Remaining 減一，並記錄 PID(0-9, A-Z)於甘特圖，如果 process 的 CPU_Burst_Remaining 等於 0，代表程序執行完畢，記錄完成時間，並計算出 Turnaround_Time, Waiting_Time，接者將完成的 process 加入至 Done_List 中，完成 SRTF_Simulate.Start()後，最後呼叫 PrintResult()將結果寫檔。

6.method 4： # PPRR (Preemptive Priority Round Robin)
首先建立 PPRR_Process_List，並複製 processList 的內容，代表 input 資訊，接者利用 class PPRR 建立 PPRR_Simulate，之後呼叫 PPRR_Simulate.Start()進行 CPU 排程，PPRR_Simulate.Start()會先依 Arrival Time 排序所有 process，如果 Arrival Time 相同，再利用 Priority 排序，還是一樣再利用 PID 排序，確認順序後執行 CheckProcess()，CheckProcess()的目的是將抵達的 process 放進 Waiting_Queue 裡面，並將放入 Waiting_Queue 的 process 從 PPRR_Process_List 中 pop 出來，代表 process 已處裡過，在這裡我會先重新排序一次，依據 process 的 Priority, Has_Use_CPU(是否曾經執行過), Arrival Time, process ID 排序，接者我會把 Waiting_Queue 中與正在執行的 process 的 Priority 相同者放入 Same_Priority_Queue，代表等一下會優先處理，因為 PPRR 是以 Priority 優先處理為原則，這裡有個重點要注意，因為 PPRR 是 Preemptive，所以如果有 process 的 Priority 小於正在執行的 process 的 Priority，就進行搶奪，原本正在執行的 process 進入 Waiting_Queue，另外，如果 Priority 相同，就比較 Arrival Time，如果還是相同，再比較 PID，如果成立，依然可以進行搶奪，接者執行 RunProcess()，RunProcess()優先從 Same_Priority_Queue pop process 出來執行，其次才是 Waiting_Queue，執行時將 process 的 CPU_Burst_Remaining 減 1，也把 Time_Slice 減 1，還有 Has_Use_CPU 設為 True，並記錄 PID(0-9, A-Z)於甘特圖，如果 process 的 CPU_Burst_Remaining 等於 0，代表程序執行完畢，記錄完成時間，並計算出 Turnaround_Time, Waiting_Time，接者將完成的 process 加入至 Done_List 中，完成 PPRR_Simulate.Start()後，最後呼叫 PrintResult()將結果寫檔。

7.method 5： # HRRN (High Response Ratio Next)
首先建立 HRRN_Process_List，並複製 processList 的內容，代表 input 資訊，接者利用 class HRRN 建立 HRRN_Simulate，之後呼叫 HRRN_Simulate.Start()進行 CPU 排程，HRRN_Simulate.Start()會先依 Arrival Time 排序所有 process，如果 Arrival Time 相同，再利用 PID

排序，確認順序後執行 CheckProcess()，CheckProcess()的目的是將抵達的 process 放進 Waiting_Queue 裡面，並將放入 Waiting_Queue 的 process 從 HRRN_Process_List 中 pop 出來，代表 process 已處裡過，另外，也要計算 Waiting_Queue 中各個 process 的 Response_Ratio，利用 Current_Time 減去 process 的 Arrival_Time 得到目前等待時間，再加上 CPU_Burst，最後再同除 CPU_Burst，所得結果即為 Response_Ratio，之後重新排序一次，依據 Response_Ratio(越大者越前)，Arrival_Time, PID，接者執行 RunProcess()，RunProcess()從 Waiting_Queue 中 pop process 出來並執行，執行時將 process 的 CPU_Burst_Remaining 減 1，並記錄 PID(0-9, A-Z) 於甘特圖，如果 process 的 CPU_Burst_Remaining 等於 0，代表程序執行完畢，記錄完成時間，並計算出 Turnaround_Time, Waiting_Time，接者將完成的 process 加入至 Done_List 中，完成 HRRN_Simulate.Start()後，最後呼叫 PrintResult()將結果寫檔。

8. method 6： # ALL
其實就是 method 1-5 的整合，依序執行上述的程式碼，最後呼叫 PrintAllResult()將結果寫檔。

三、使用的 Data Structure

1. Process：

```
3   class Process() :
4     def __init__( self, id, CPU_burst, arrivalTime, priority ) :
5       self.ID = id
6       self.CPU_Burst = CPU_burst
7       self.Arrival_Time = arrivalTime
8       self.Priority = priority
9
10      self.Complete_Time = 0
11      self.Waiting_Time = 0
12      self.Turnaround_Time = 0
13
14      self.Time_Slice = 0  # RR, PPRR
15      self.CPU_Burst_Remaining = 0
16      self.Time_Slice_Limit = False  # PPRR
17      self.Has_Use_CPU = False  # PPRR
18      self.Response_Ratio = 0.0  # HRRN
19
```

前 4 項為讀檔而來的資料，型別是 integer，中間 3 項為記錄結果之用途，型別是 integer，後面數項是特別處理各方法時所需的資料，例如：在 RR, PPRR 時會需要用到 Time_Slice，型別是 integer，

CPU_Burst_Remaining 在各方法皆需要用到，目的是在記錄該 process 還剩下多少作業時間，特別是在 SRTF 時，會依此作為優先排序依據，型別是 integer，在 PPRR 中我會利用 Time_Slice_Limit 代表是否受到 Time_Slice 的限制，因為如果 Priority 較小時，可優先處理並且不受 RR 限制，型別是 boolean，Response_Ratio 就是 HRRN 排序時的依據，比率越高可優先處理，型別是 float。

2.FCFS：

```
20  class FCFS() :
21    def __init__( self, processList ) :
22      self.Method_Name = "FCFS"
23      self.Process_List = processList
24      self.Gantt_Chart = ""
25      self.Running_Process = None
26      self.Waiting_Queue = []
27      self.Done_List = []
28      self.Process_Quantity = len( processList )
29      self.Current_Time = 0
```

Method_Name 用來記錄方法的名稱，型別是 string，Process_List 是讀入的內容，型別是 list，Gantt_Chart 用來記錄甘特圖，型別是 string，Running_Process 代表正在執行的 process，型別是 Process，Waiting_Queue 代表等待的佇列，裡面存的是 process，型別是 list，Done_List 代表完成的佇列，裡面存的是 process，型別是 list，Process_Quantity 用來記錄 input 的數量，型別是 integer，Current_Time 代表此時程式運行的時間，型別是 integer。

3.RR：

```
77  class RR() :
78    def __init__( self, processList, timeSlice ) :
79      self.Method_Name = "RR"
80      self.Process_List = processList
81      self.Time_Slice = timeSlice
82      self.Gantt_Chart = ""
83      self.Running_Process = None
84      self.Waiting_Queue = []
85      self.Done_List = []
86      self.Process_Quantity = len( processList )
87      self.Current_Time = 0
```

Method_Name 用來記錄方法的名稱，型別是 string，Process_List 是讀入的內容，型別是 list，Time_Slice 型別是 integer，Gantt_Chart 用來記錄甘特圖，型別是 string，Running_Process 代表正在執行的 process，型別是 Process，Waiting_Queue 代表等待的佇列，裡面存的

是 process，型別是 list，Done_List 代表完成的佇列，裡面存的是 process，型別是 list，Process_Quantity 用來記錄 input 的數量，型別是 integer，Current_Time 代表此時程式運行的時間，型別是 integer。

4.SRTF：

```
151 class SRTF() :
152   def __init__( self, processList ) :
153     self.Method_Name = "SRTF"
154     self.Process_List = processList
155     self.Gantt_Chart = ""
156     self.Running_Process = None
157     self.Waiting_Queue = []
158     self.Done_List = []
159     self.Process_Quantity = len( processList )
160     self.Current_Time = 0
```

Method_Name 用來記錄方法的名稱，型別是 string，Process_List 是讀入的內容，型別是 list，Gantt_Chart 用來記錄甘特圖，型別是 string，Running_Process 代表正在執行的 process，型別是 Process，Waiting_Queue 代表等待的佇列，裡面存的是 process，型別是 list，Done_List 代表完成的佇列，裡面存的是 process，型別是 list，Process_Quantity 用來記錄 input 的數量，型別是 integer，Current_Time 代表此時程式運行的時間，型別是 integer。

5.PPRR：

```
233 class PPRR() :
234   def __init__( self, processList, timeSlice ) :
235     self.Method_Name = "PPRR"
236     self.Process_List = processList
237     self.Time_Slice = timeSlice
238     self.Gantt_Chart = ""
239     self.Running_Process = None
240     self.Waiting_Queue = []
241     self.Done_List = []
242     self.Process_Quantity = len( processList )
243     self.Current_Time = 0
244     self.Same_Priority_Queue = []
```

Method_Name 用來記錄方法的名稱，型別是 string，Process_List 是讀入的內容，型別是 list，Time_Slice 型別是 integer，Gantt_Chart 用來記錄甘特圖，型別是 string，Running_Process 代表正在執行的 process，型別是 Process，Waiting_Queue 代表等待的佇列，裡面存的

是 process，型別是 list，Done_List 代表完成的佇列，裡面存的是 process，型別是 list，Process_Quantity 用來記錄 input 的數量，型別是 integer，Current_Time 代表此時程式運行的時間，型別是 integer，Same_Priority_Queue 代表有相同 priority 的 process 的佇列，型別是 list。

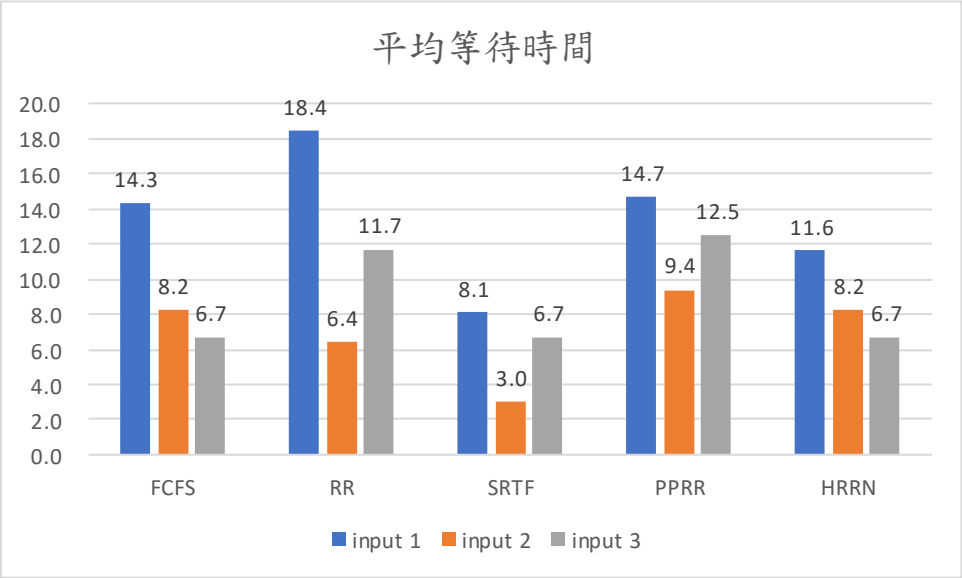
6.HRRN：

```
354 class HRRN() :
355     def __init__( self, processList ) :
356         self.Method_Name = "HRRN"
357         self.Process_List = processList
358         self.Gantt_Chart = ""
359         self.Running_Process = None
360         self.Waiting_Queue = []
361         self.Done_List = []
362         self.Process_Quantity = len( processList )
363         self.Current_Time = 0
```

Method_Name 用來記錄方法的名稱，型別是 string，Process_List 是讀入的內容，型別是 list，Gantt_Chart 用來記錄甘特圖，型別是 string，Running_Process 代表正在執行的 process，型別是 Process，Waiting_Queue 代表等待的佇列，裡面存的是 process，型別是 list，Done_List 代表完成的佇列，裡面存的是 process，型別是 list，Process_Quantity 用來記錄 input 的數量，型別是 integer，Current_Time 代表此時程式運行的時間，型別是 integer。

四、不同排程法的比較

1.平均等待時間：



2.結果與討論:

上圖為 3 筆測資的結果，整體而言 SRTF 的表現最好，其次是 HRRN 及 FCFS，最後是 RR 及 PPRR，可以觀察到上述 2 種方法浮動性較大，因為 Time Slice 的設定很重要，在 input 1 中，Time Slice 為 1，在執行 RR 及 PPRR 時效果不佳，執行起來像是 FCFS 一樣，以穩定性來說，我覺得 HRRN 最好，透過 Response Ratio 的計算，也可以平衡 process 的等待時間，最不推薦 PPRR，雖然有 RR 做限制，但是只要 priority 較高者，還是有機會霸占整個程序執行空間。

五、未完成的功能

無。