Operating System Project2

CPU Scheduling

資訊三乙 10727211 林彥輝

一、開發環境:

作業系統: Windows 10 Enterprise x64

使用語言: Python 3.8

測試環境:

- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (12 CPUs),~2.6GHz
- RAM: Micron DDR4 2666/32GB*1
- SDD: 1TB PCIe SSD
- OS: Windows 10 Enterprise x64
- IDE: PyCharm Community Edition 2020.2.2

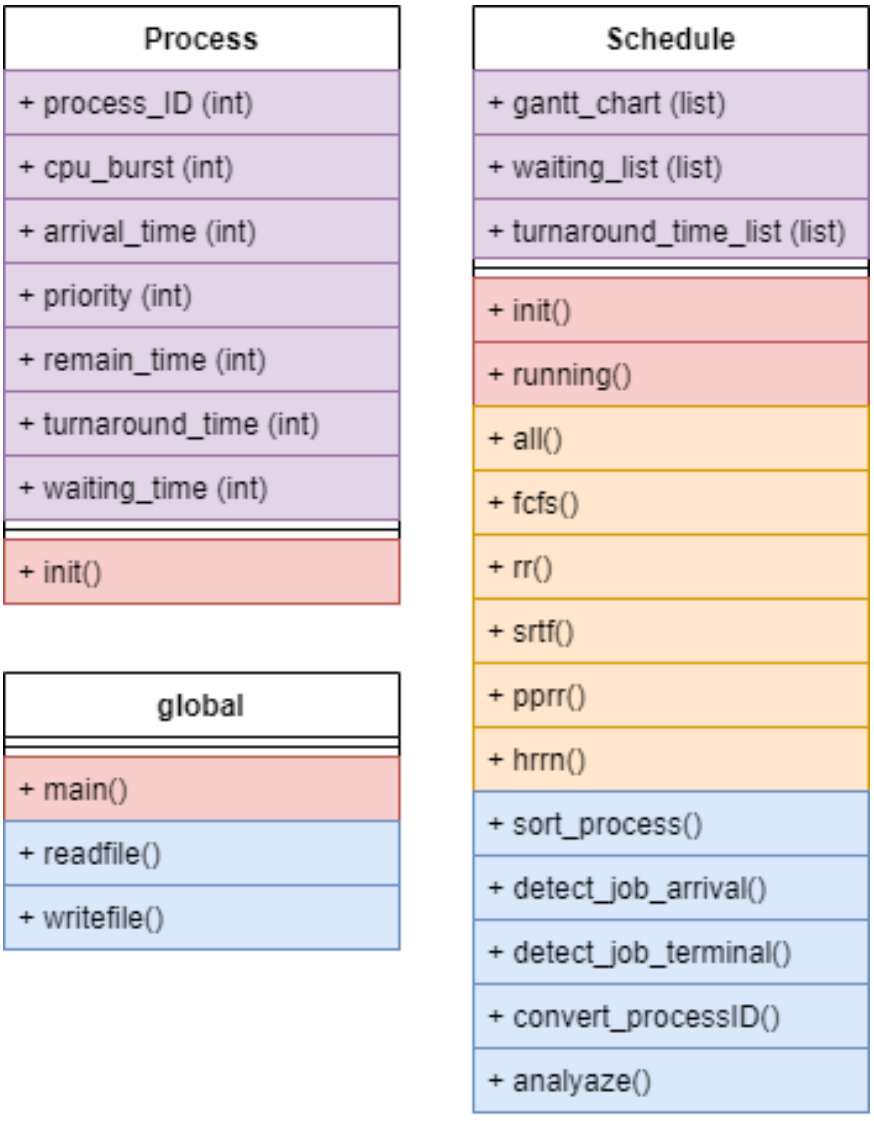
二、實作功能

本次 Project 中實作以下 CPU Scheduling 之功能：

1. First Come First Serve (FCFS)
2. Round Robin (RR)
3. Shortest Remaining Time First (SRTF)
4. Preemptive Priority + Round Robin (PPRR)
5. Highest Response Ratio Next (HRRN)
6. Run All of Above in a time (1-5)

三、實作流程

實作模擬 CPU Scheduling 之 Data Structure 如下：



※正式名稱與變數名稱格式為：變數名稱(正式名稱)。

※以下解說將統一使用正式名稱說明。

Class Process：模擬一個 Job 的所有資訊，包括 process_ID(ID)、cpu_burst(CPU Burst)、arrival_time(Arrival Time)、priority(Priority)，以及更進階的資訊，包括 remain_time(Remaining Time)、turnaround_time(Turnaround Time)、waiting_time(Waiting Time)，這些欄位提供 Object 在排程計算時有良好的評斷比較。

Class Schedule：提供一次排程所需的所有計算 Function，也將最後的計算結果放入 gantt_char(甘特圖)、waiting_list、turnaround_time_list 中。

- running() ： 為 Schedule 的核心 Function，根據讀入檔案所需指令呼叫 all()、fcfs()、rr()、srtf()、pprr()、hrrn()，並呼叫 analyaze()統計 waiting time 及 turnaround time。

- all():為方法六之對應 Function，呼叫方法一到方法五的 Function，且蒐集各方法中的數據。

- fcfs():為方法一之對應 Function，使用一維陣列 ready_queue (Ready Queue)、Object running_queue(Running State)、一維陣列 terminate_queue(Terminated)做為排程工具，先將已抵達的 Ready Queue 的 Process 透過 sort_process()先進行 Process ID 排序，再透過 arrival time 排序，即可順利抓取題目要求順位（Arrival Time 越低 >> Process ID 越小)之 Process 放入 Ready Queue 執行，最後將 Running State 中 remaining time 為 0 的 process 放入 Terminated 中，過程中順帶紀錄該方法之甘特圖。

- rr():為方法二之對應 Function，使用一維陣列 ready_queue (Ready Queue)、Object running_queue(Running State)、一維陣列 terminate_queue(Terminated)做為排程工具，將已抵達的 Ready Queue 的 Process 透過 sort_process()先進行 Process ID 排序，再透過 arrival time 排序，放入 running State 執行，過程中發生 Timeout 將會進行 content switch，分派下一個 Ready queue 的 Process 進 Running State，最後將 Running State 中 remaining time 為 0 的 process 放入 terminated 中，過程中順帶紀錄該方法之甘特圖。

- srtf():為方法三之對應 Function，使用一維陣列 ready_queue (Ready Queue)、Object running_queue(Running State)、一維陣列 terminate_queue(Terminated)做為排程工具，將已抵達的 Ready Queue 的 Process 透過 sort_process()依序針對 Process ID、arrival time、remaining time 排序，放入 Running State 執行，過程中發生可能根據 remaining time 進行 preemptive，分派下一個 Ready Queue 的 Process 進 Running State，最後將 Running State 中 remaining time

為 0 的 Process 放入 Terminated 中，過程中順帶紀錄該方法之甘特圖。

- pprr():為方法四之對應 Function，使用一維陣列 ready_queue (Ready Queue)、Object running_queue(Running State)、一維陣列 terminate_queue(Terminated)做為排程工具，先將已抵達的 Ready Queue 的 Process 透過 sort_process()依序針對 arrival time、Priority 排序，放入 Running Queue 執行，過程中發生根據 Timeout 進行 content switch，分派下一個同等 Priority 的 Process 進 Running Queue，亦或是有更高的 Process 進入 Ready Queue，則會立即遭到 Preemptive 而非 Timeout，最後將 Running Queue 中 remaining time 為 0 的 Process 放入 terminated 中，過程中順帶紀錄該方法之甘特圖。

- hrrn():為方法五之對應 Function，使用一維陣列 ready_queue (Ready Queue)、Object running_queue(Running State)、一維陣列 terminate_queue(Terminated)做為排程工具，先將已抵達的 Ready Queue 的 Process 透過 sort_process()依序針對 Process ID、arrival time、response_ratio 排序，response_ratio 計算的方法為(time – arrival time + remaining time) / CPU Burst，並覆蓋原先 Process 中存放 Priority 的欄位，放入 Running Queue 執行，最後將 Running Queue 中 remaining time 為 0 的 Process 放入 Terminated 中，過程中順帶紀錄該方法之甘特圖。

- sort_process(): 排序方法為 Insertion sort，為了資料的穩定性，且有助於資料比對的順位。如若先比較 Arrival Time，Arrival Time 相同時比較 Process ID，則可先呼叫 sort_process()進行 Process ID 的排序，再做 Arrival Time 的排序，即可確保排序的順序。

- detect_job_arrival():檢查 Process 是否抵達(根據 Arrival Time)。

- detect_job_terminal():檢查 Process 是否完成(根據 remaining Time)。

- convert_processID():在製作甘特圖時，將 Process ID 轉換成對應的代號。

- analyaze():透過 Terminated 中的 waiting_list、turnaround_time_list 進行二維資料的蒐集，以便輸出檔案。蒐集結果存放如下圖。

```
['waiting']
['ID', 'FCFS', 'RR', 'SRTF', 'PPRR', 'HRRN']
['============================================================']
[1, 0, 0, 0, 0, 0]
[2, 0, 20, 0, 30, 0]
[3, 20, 30, 20, 35, 20]
[4, 15, 15, 15, 0, 15]
[5, 0, 0, 0, 10, 0]
[6, 5, 5, 5, 0, 5]
['============================================================']
```

```
['Turnaround Time']
['ID', 'FCFS', 'RR', 'SRTF', 'PPRR', 'HRRN']
['============================================================']
[1, 20, 20, 20, 20, 20]
[2, 25, 45, 25, 55, 25]
[3, 45, 55, 45, 60, 45]
[4, 30, 30, 30, 15, 30]
[5, 10, 10, 10, 20, 10]
[6, 15, 15, 15, 10, 15]
['============================================================']
```

四、困難處與心得

在 C 中有 call by value、call by reference 的概念，轉換至 Python 後卻發現相同的思路卻因不同的語言有不同的結果，如 list 的 copy[1]，有 deep copy(call by value in Clang)、Shallow copy(call by reference in Clang)，上網稍微翻閱了資料才弄懂兩個語言之間需要補充的小知識，也對 Python 有了更進一步的接觸。

這次的 CPU Scheduling 是一個相當有趣的作業，有效率的排程看似簡單，如 SJF、SRTF(本次實作)，但實質上相當難實作，因為無法精準的預判一

個 Job 所需 CPU Burst 會是多少，寫報告的同時回頭觀察這些排程法，會發現 HRRN 會小優於 PPRR 與 RR，但效率都輸給 SRTF 一大截，要如何將演算法進步至 RR，結合現階段學到各演算法的優點，且同時要考慮到 Starving 問題，是一個相當大的學問，透過這次的作業，除了碰觸 Python 之類不熟悉的語言之外，也更進一步的了解到計算機的世界，受益良多！

五、參考資料

1. **Deep Copy & Shallow Copy in Python**