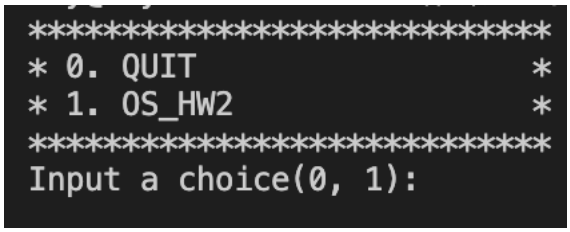
作業系統作業二 * 有做流程圖

資訊三乙 10833230 李郁含

1.開發環境：MacBook air 2020 M1 / Visual Studio Code / 使用語言：Python

2.實作方法和流程: 開始執行會出現以下畫面輸入



1 能不斷重複執行流程

直到輸入 0 退出

便開始整個作業的輸入流程，首先輸入檔案名稱，將檔案讀進後切出第一行第一個數字為 method，第二個數字為 time slice，最後將數字全部增加進 list。

將 list 裡的每一個分四個一組一組丟進 process 建立 process 資料，首先先將 list 以 process ID 先做排序，依照使用者輸入的 method 做不一樣的排程

如果使用者輸入 1 或 6 便進行 FCFS

如果使用者輸入 2 或 6 便進行 RR

如果使用者輸入 3 或 6 便進行 SRTF

如果使用者輸入 4 或 6 便進行 Priority RR

如果使用者輸入 5 或 6 便進行 HRRN

依照使用者輸入的方法寫檔不同的格式，並也將進行各種排程時紀錄的甘特圖、waiting_time 和 turnaround_time 依照使用者輸入的方法寫檔

以使用者輸入的 method 來決定要執行的 排程方法:

FCFS： 依 ArrivalTime 先後次序進行排程

將 list deepcopy 一份（因為這樣如果 list 改變，複製品 list_FCFS 就不會跟著改變），接著宣告一個繼承 schedule 的 FCFS class 便開始進行資料的排程。宣告一個無限容量的 queue 放等待中的 process 和一個初始值為 None 的 process 放正在 running 的 process，進入一個迴圈（條件是只要 readyqueue 還不是空的或還有一個 process 還沒做完），找到此刻到達的 process 們並丟進 ready queue，判斷 running process 是否為 None (是否有正在執行的 process)，(１) 如有：先讓他執行完一次並同時更新存在 schedule 裡的甘特圖，再判斷 process 是否已經執行完，如執行完了，便將完成的 process 加入到 donelist 並同時設定目前沒有 process 在執行；(２) 如沒有（有兩種可能：還未有 process 進去或 process 皆已做完）：先查看 readyqueue 有沒有 process 在等待，如有便從 readyq 裡 get 一個 process 出來，如沒有且確定不是結束(process 皆已做完)，代表目前這段時間不會有 process 執行便更新存在 schedule 裡的甘特圖（老師要

求的 - ），最後將目前時間增加 1

RR ： 依 ArrivalTime 先後次序進行排程 加上 timeslice 的限制

將 list deepcopy 一份（因為這樣如果 list 改變，複製品 list_RR 就不會跟著改變），接著宣告一個繼承 schedule 的 RR class 便開始進行資料的排程。宣告一個無限容量的 queue 放等待中的 process 和一個初始值為 None 的 process 放正在 running 的 process，進入一個迴圈（條件是只要 readyqueue 還不是空的或還有一個 process 還沒做完），再宣告一個 count 暫時代替 timeslice，並再進入一個迴圈，讓我們能知道在排程過程 process 是否 time out 了（count 代替 timeslice 扣除以計算）， 找到此刻到達的 process 們並丟進 ready queue，判斷 running process 是否為 None (是否有正在執行的 process)，（1）如有：先讓他執行完一次並同時更新存在 schedule 裡的甘特圖，再判斷 process 是否已經執行完，如執行完了，便將完成的 process 加入到 donelist 並同時設定目前沒有 process 在執行；（2） 如沒有（有兩種可能：還未有 process 進去或 process 皆已做完):先判斷是否 timeout 了！如沒有先查看 readyqueue 有沒有 process 在等待，如有便從 readyq 裡 get 一個 process 出來，如沒有且確定不是結束 (process 皆已做完)，代表目前這段時間不會有 process 執行便更新存在 schedule 裡的甘特圖（老師要求的 - ）
(3)如果 time out 了！將還未完成的 process 推進 readyq，並同時從 readyq 裡 get 出一個 process 當作正在執行的 process，最後將目前時間增加 1，而同時 count 也會減少

SRTF ： 依 ArrivalTime 先後次序進行排程 但每次剩餘 CPU burst 較小的先（沒有 timeslice 的限制)

將 list deepcopy 一份（因為這樣如果 list 改變，複製品 list_SRTF 就不會跟著改變），接著宣告一個繼承 schedule 的 SRTF class 便開始進行資料的排程。宣告一個無限容量的 Heap 放等待中的 process，和一個初始值為 None 的 process 放正在 running 的 process，進入一個迴圈(條件是只要 readyheap 還不是空的或還有一個 process 還沒做完)，找到此刻到達的 process 們並丟進 ready heap（!!注意這裡更新 ready 的方法是覆寫 schedule 的，因為 SRTF 前面宣告的是 heap，推進 queue(put)和推進 heap(push)的使用方法不同），判斷 running process 是否為 None (是否有正在執行的 process)，（1）如有：先讓他執行完一次並同時更新存在 schedule 裡的甘特圖，再判斷 process 是否已經執行完，如執行完了，便將完成的 process 加入到 donelist 並同時設定目前沒有 process 在執行；(2) 如沒有(有兩種可能:還未有 process 進去或 process 皆已做完）：先查看 readyqueue 有沒有 process 在等待，如有便從 readyq 裡 get 一個 process 出來，如沒有且確定不是結束(process 皆已做完)，代表目前這段時間不會有 process 執行便更新存在 schedule 裡的甘特圖(老師要求的 -); (3) 如有且隨著 minheap（key : burst)變動，只要有比現在在 running 的 process 之 burst 還要小的話就停止執行此 process，並換上新的（更小的），最後將目前時間增加 1

Priority RR ： 依 ArrivalTime 先後次序進行排程 但每次 priority 較小的先（有

timeslice 的限制)

將 list deepcopy 一份（因為這樣如果 list 改變，複製品 list_PPRR 就不會跟著改變），接著宣告一個繼承 schedule 的 PPRR class 便開始進行資料的排程。宣告一個無限容量的 Heap 放等待中的 process，和一個初始值為 None 的 process 放正在 running 的 process，進入一個迴圈（條件是只要 readyheap 還不是空的或還有一個 process 還沒做完），再宣告一個 count 暫時代替 timeslice，並再進入一個迴圈，讓我們能知道在排程過程 process 是否 time out 了（count 代替 timeslice 扣除以計算），找到此刻到達的 process 們並丟進 ready heap，判斷 running process 是否為 None (是否有正在執行的 process)，（1）如有：先讓他執行完一次並同時更新存在 schedule 裡的甘特圖，再判斷 process 是否已經執行完，如執行完了，便將完成的 process 加入到 donelist 並同時設定目前沒有 process 在執行；（2）如沒有(有兩種可能：還未有 process 進去或 process 皆已做完）：先查看 readyqueue 有沒有 process 在等待，如有便從 readyq 裡 get 一個 process 出來，如沒有且確定不是結束(process 皆已做完)，代表目前這段時間不會有 process 執行便更新存在 schedule 裡的甘特圖（老師要求的 - ）;（3）如有且隨著 minheap（key : burst)變動，只要有比現在在 running 的 process 之 burst 還要小的話或 time out 了！！就停止執行此 process，並換上新的（更小的），最後將目前時間增加 1，而同時 count 也會減少

HRRN ：依 ArrivalTime 先後次序進行排程 但每次反應時間比率較大的先（沒有 timeslice 的限制)

將 list deepcopy 一份（因為這樣如果 list 改變，複製品 list_HRRN 就不會跟著改變），接著宣告一個繼承 schedule 的 HRRN class 便開始進行資料的排程。宣告一個無限容量的 queue 放等待中的 process 和一個初始值為 None 的 process 放正在 running 的 process，進入一個迴圈（條件是只要 readyqueue 還不是空的或還有一個 process 還沒做完），找到此刻到達的 process 們並丟進 ready queue，判斷 running process 是否為 None (是否有正在執行的 process)，（1）如有 : 先讓他執行完一次並同時更新存在 schedule 裡的甘特圖，再判斷 process 是否已經執行完，如執行完了，便將完成的 process 加入到 donelist 並同時設定目前沒有 process 在執行，接著宣告一個宣告一個暫時無限容量的 Heap 讓 ready 裡面的 data 以 key 值為反應時間比率做 reheap(max_heap)

HRRN maxheap:將 readyqueue 裡面的 data 全拿出來丟進 maxheap 一個一個更新反應時間比率，每一次去找最大的反應時間比率

反應時間比率的計算：是將目前時間減掉起出的抵達時間在除以 cpu 的執行時間（burst）

將暫放的 list 先以 process ID 先做排序，再用一個迴圈一個一個推進原本的 queue

；（2）如沒有(有兩種可能 : 還未有 process 進去或 process 皆已做完）: 先查看 readyqueue 有沒有 process 在等待，如有便從 readyq 裡 get 一個 process 出來，如沒有且確定不是結束(process 皆已做完)，代表目前這段時間不會有 process 執行便更新存在 schedule 裡的甘特圖（老師要求的 - ），最後將目前時間增加 1

3. 不同排程法的比較: 舉例 方法 : 6 、timeslice : 3 （ input2 ）

Waiting Time

ID	FCFS	RR	SRTF	PPRR	HRRN
1	0	13	13	0	0
2	10	2	0	21	10
3	10	2	0	8	12
4	11	6	1	9	8
5	11	9	1	9	11
Average Waiting Time	8.4	6.4	3	9.4	8.2

Turnaround Time

ID	FCFS	RR	SRTF	PPRR	HRRN
1	11	24	24	11	11
2	12	4	2	23	12
3	13	5	3	11	15
4	13	8	3	11	10
5	17	15	7	15	17
Average Turnaround Time	13.2	11.2	7.8	14.2	13

4. 結果與討論

SRTF 不只是 waiting time 普遍花的時間最少，因為他是可奪取的且隨時注意是否有較小使用 CPU 時間之 process 出現，因此他有最小的平均等待時間，所以如果是以等待時間來評斷排程方法的好壞的話，那麼 SRTF 是最好的演算法。

FCFS： non-preemptive

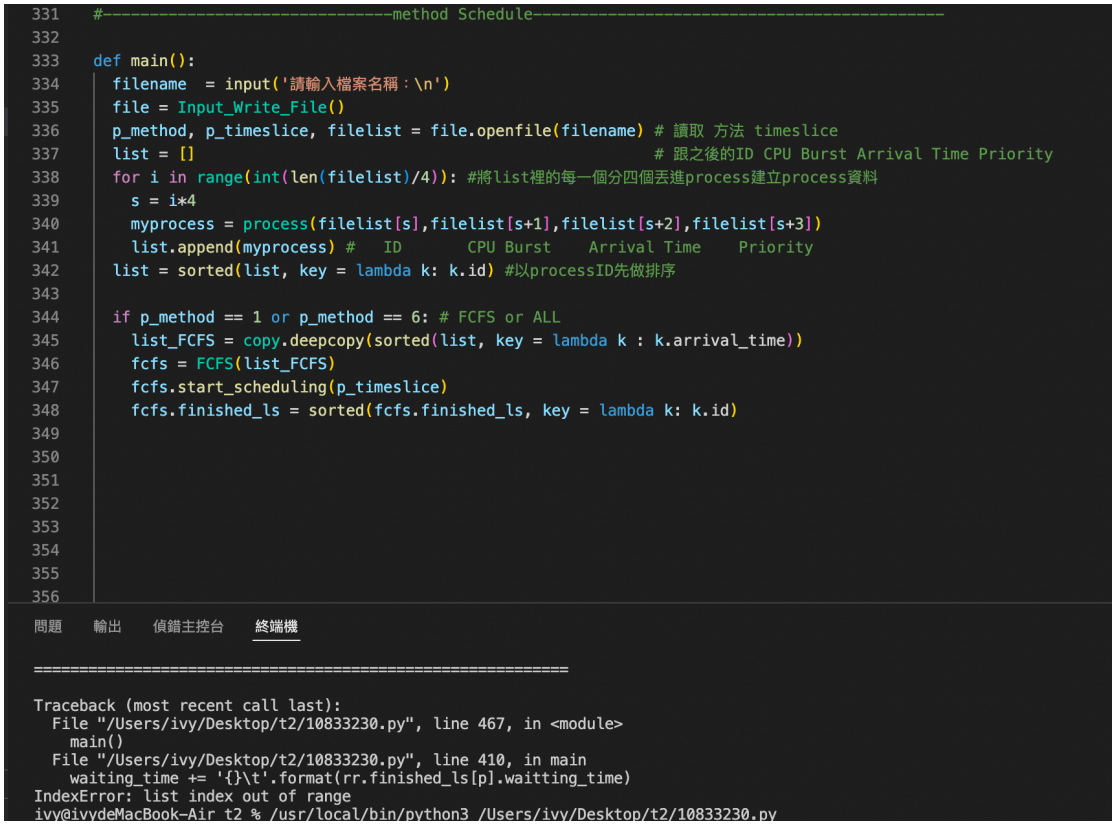
RR ： preemptive

SRTF ： preemptive

Priority RR ： preemptive

HRRN ： non-preemptive

遇到的問題以及解決方法：



```
331  #------------------------------method Schedule-----------------------------------------
332
333  def main():
334    filename  = input('請輸入檔案名稱：\n')
335    file = Input_Write_File()
336    p_method, p_timeslice, filelist = file.openfile(filename) # 讀取 方法 timeslice
337    list = []                                                 # 跟之後的ID CPU Burst Arrival Time Priority
338    for i in range(int(len(filelist)/4)): #將list裡的每一個分四個丟進process建立process資料
339      s = i*4
340      myprocess = process(filelist[s],filelist[s+1],filelist[s+2],filelist[s+3])
341      list.append(myprocess) #   ID       CPU Burst    Arrival Time    Priority
342    list = sorted(list, key = lambda k: k.id) #以processID先做排序
343
344    if p_method == 1 or p_method == 6: # FCFS or ALL
345      list_FCFS = copy.deepcopy(sorted(list, key = lambda k : k.arrival_time))
346      fcfs = FCFS(list_FCFS)
347      fcfs.start_scheduling(p_timeslice)
348      fcfs.finished_ls = sorted(fcfs.finished_ls, key = lambda k: k.id)
349
350
351
352
353
354
355
356
```

問題　輸出　偵錯主控台　終端機

```
============================================================

Traceback (most recent call last):
  File "/Users/ivy/Desktop/t2/10833230.py", line 467, in <module>
    main()
  File "/Users/ivy/Desktop/t2/10833230.py", line 410, in main
    waiting_time += '{}\t'.format(rr.finished_ls[p].waitting_time)
IndexError: list index out of range
ivy@ivydeMacBook-Air t2 % /usr/local/bin/python3 /Users/ivy/Desktop/t2/10833230.py
```

第 345 和 346 行，我本來是直接將排序完的 list 丟進 FCFS class 便出現 list index out of range 後來改增加了一行 將原本的 list deepcopy 給一個新的 list 就解決了！

推測應該是，由於 python 中賦值 b=a，將 b 指向了與 a 的同一個 list，當更改了 a 列表中的元素，b 再引用就會出現【list index out of range】的錯誤。

因此將 list deepcopy 一份（因為這樣如果 list 改變，複製品 list_FCFS 就不會跟著改變）

前面 method 不斷提到