OS Project2 書面報告

資訊三甲 11027164 趙怡儒

1. 開發環境 : Visual Studio Code 使用 Python
2. 實作方法和流程 :

先將input檔輸入，存成dict的資料結構，並且存入時先用'ArrivalTime'由小到大排列，若有相同'ArrivalTime'，則再以'ID'排列。

再用data['method']判斷要用哪種方法執行。

1. FCFS : FCFS如同名字一般，先到的先服務而且不能奪取，所以排好的dict可以直接操作，唯一要注意的點是如果目前時間下一個process還沒進來，就要印出-表CPU目前idle，然後直接將目前時間跳到下一個process的抵達時間。
2. RR:將已抵達的process放到等待佇列裡面，因為佇列是FIFO的，所以每次都拿第一個出來執行，比較它的CPU Burst 與 time slice ，如果CPU Burst比較大，就執行一個time slice的時間然後回去排隊，但回去排隊前要先檢查這段時間是否有新的process來了，有的話就要先讓新來的排前面。每次迴圈開始前都檢查有沒有process到了然後還沒在等待佇列裡面。
3. SJF: 將已抵達的process放到等待list裡面，這個方法用list是因為它需要比較目前可執行的process中，誰的CPU Burst最小就先執行誰，且因為不能奪取，一次執行完它的CPU Burst。
4. SRTF:與SJF相似，都是看誰的CPU Burst最小就先執行誰，但此功能是看”剩餘”的CPU Burst，而且是可以奪取的，所以我採用的方式是讓目前時間每次加一個時間單位，每個時間單位都比較誰的剩餘CPU Burst比較少。
5. HRRN:也是用list為主要等待的資料結構，因為這個方法主要是比較每個process的反應時間比率，反應時間比率= ( ( 目前時間– 抵達時間) + CPU Burst ) / CPU Burst，不能奪取，所以每次執行就直接執行完此process的CPU Burst。
6. PPRR:將已抵達的process按照Priority排序，若遇到相同Priority就變成RR，跟SRTF一樣可奪取，所以一樣採用目前時間每次加一個時間單位，每個時間單位都比較誰的Priority比較小，如果執行期間遇到比較優先的，則break while迴圈，同時讓原本在執行的process回去排隊，但回去排隊前跟RR一樣，要看有沒有新來的。
7. 補充說明：其他function或每個排程法一樣的地方

(1).每個排程法的迴圈終止條件都是每個process['CPUBurst'] 都等於 0了，意即每個process都做完了

但一開始發現如果直接用data的值去改的話，最一開始讀進來的值也會被改掉，所以後來改用



在每個方法開始前都先另一份資料讓我改動，這樣不僅可以修改數值讓我的條件成立，還可以保留原本的數值。

(2). 

兩個function是我在需要rr排程法時，確保沒有人重覆排隊。

(3). 一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

在每個排程法中，當process的CPU Burst等於0代表它完成時，就會將它的id,waiting time (= current time - process ['Arrival\_Time'] -process['CPUBurst'] )與turnaround\_time (= current\_time - process ['ArrivalTime'])記錄到process\_list中，每個排程法都有自己的process\_list。

(4).因為每個排程法都分別記錄，到要輸出的時候就特別麻煩，所以一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

另外寫了一個funtion把值存成

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

包含process id 與此process在各個方法中的waiting time的list，同理將turnaround time也做一樣的整理。

1. 不同排程法之比較

平均等待時間(Average Waiting Time)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | FCFS | RR | SJF | SRTF | HRRN | PPRR |
| Input1 | 14.333 | 18.4 | 8.866 | 8.066 | 11.6 | 14.666 |
| Input2 | 8.4 | 6.4 | 8.2 | 3 | 8.2 | 9.4 |
| Input3 | 6.667 | 11.667 | 6.667 | 6.667 | 6.667 | 12.5 |
| Input4 | 3.75 | 5.5 | 3.5 | 3.25 | 3.75 | 4.5 |

平均往返時間(Average Turnaround Time)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | FCFS | RR | SJF | SRTF | HRRN | PPRR |
| Input1 | 18.2 | 22.266 | 12.733 | 11.933 | 15.466 | 18.533 |
| Input2 | 13.2 | 11.2 | 13 | 7.8 | 13 | 14.2 |
| Input3 | 24.166 | 29.166 | 24.166 | 24.166 | 24.166 | 30 |
| Input4 | 8.75 | 10.5 | 8.5 | 8.25 | 8.75 | 9.5 |

四ˋ 結果與討論

1.只要有用到RR的等待時間跟往返時間大部分都高於其他人，合理推論應該是因為RR做完一個time slice後就要重新排隊，如果排隊的process很多的話，那兩種時間都會越長，所以兩種時間的長短跟排隊的process與time slice的長度應該有密切的關係。

2. SJF與SRTF的等待時間跟往返時間大部分都低於其他人，合理推論應該是因為把要執行時間比較久的process都丟到後面去了，雖然其process本人的兩種時間都會特別長，但是表格是取平均值，讓平均整體減少了。

3. FCFS的等待時間跟往返時間看起來是取決於前幾個到達的process的執行時間，如果前幾個到達的執行時間就特別長的話，後面到達的process的兩種時間也會特別長，平均下來數值就大了。

4.HRRN的等待時間跟往返時間都略大於SJF與SRTF的，合理推測是因為HRRN是以反應時間比率來調整順序，所以等待時間就不會被拉得太長。

5.如果比較全部的方法，FCFS如果每個process的執行時間都很小的話效率應該會不錯。RR雖然很公平公正，但time slice大小的設定是一門很深的學問，太大就等於FCFS，太小在實際上可能一直在做context switch。SJF與SRTF雖然看起來最棒，但實際上CPU不一定可以知道每個process的執行時間，所以感覺很難實現。HRRN也跟SJF與SRTF有一樣的問題，看起來很棒，但實際操作上可能會有問題，PPRR則是在實際面上最有效，畢竟判斷依據的是優先程度，讓比較重要的process先執行完。