OS Homework2

資訊三乙 11027209 巫年巨

開發環境 : python , 版本3.10.7

實作方法與流程

這次有作業要實做6個排程法與1個呼叫全部的方法，由於有大量的資料需要紀錄，而且需要把這些資料做排序，所以這次大部分的資料結構是用到queue和priority queue與array這種來去做到排程，這些要求我是用python的list來完成所有的任務，因為list就像C++的vector，既可以是1維或是多維的array，有基本的pop,append，我覺得最好用的是sort這個方法(可用lambda把二維list依照裡面的一維元素的順序來做sort，比如有存id,burst,priority,arrival的list就可以直接用sort依照id來排序這個二維list)，再來就是這次六個排程的判斷，為了接下來的題目好判斷，我先把list按照id來排序，並創了很多個list存算完的turnaround time, waiting time等等，Waiting time需要CPU Burst來計算其時間，但是我們也要把CPU Burst用完(從原本的值到0)，才能確定該id不需再用CPU，但用完就不知道原來的CPU Burst是多少了，為了解決這個問題所以在創一個CPU Burst List的紀錄現在每個ID還需用多少CPU，要使接下來排程法易於觀察，我把這個存所有東西的list先按照id來排序。

那麼就來到第一個排程法FCFS是最簡單的判斷，只要把原來的List在按照arrival time排序，然後就可以依序放入queue中，而這次任務用了clock去計算每一秒，因為要判斷該id抵達了沒，沒抵達的話要用--表達,至於id從數字轉成a-z就用一個list按照數字大小，變成index，就可以轉換了，故當時間到了在queue裡面的元素的arrival time的時候，就可以開始處理把其burst用完，把queue裡面的元素處理完並用其finish time來計算turnaround time和waiting time後就結束了。

接下來到第二個RR排程法，這個比較需要技巧一下，我的全部的完成條件是把CPU\_burst\_list的所有元素都跑完(從原本值到0)為止，然後用一個queue去存到來的id，要進去queue的條件是不在queue且已抵達又burst\_list大於0，我們每次都流程是從queue拿第0項，然後跑它的timeslice，跑完但是還有得跑的話就pop掉在append到最後面去，如果是把burst用完就pop掉，中途遇到抵達可放入的id要放進去，這樣跑完全部id就結束了。

第三個SJF到第六個PPRR都是用把CPU\_burst\_list全部用完才能結束的理念，且都是用queue存id有抵達放入，沒抵達且當下沒id的話就印-號跟continue，這個SJF排程只要比較像是PriorityQueue的理念，我是用一個inf值(無限大)和index，去每一輪去這個PriorityQueue找最小的burst，並更新index，並找題目說的burst相同找arrival time小的，如果都相同的話再去比ID，選出最小的從PriorityQueue裡面pop掉，把那個ID burst花完，持續重複到把每個ID用完後就結束了。

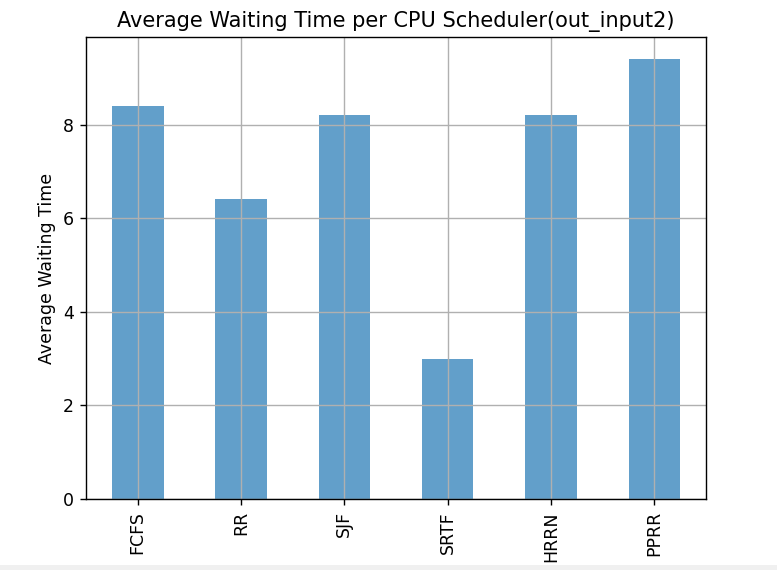
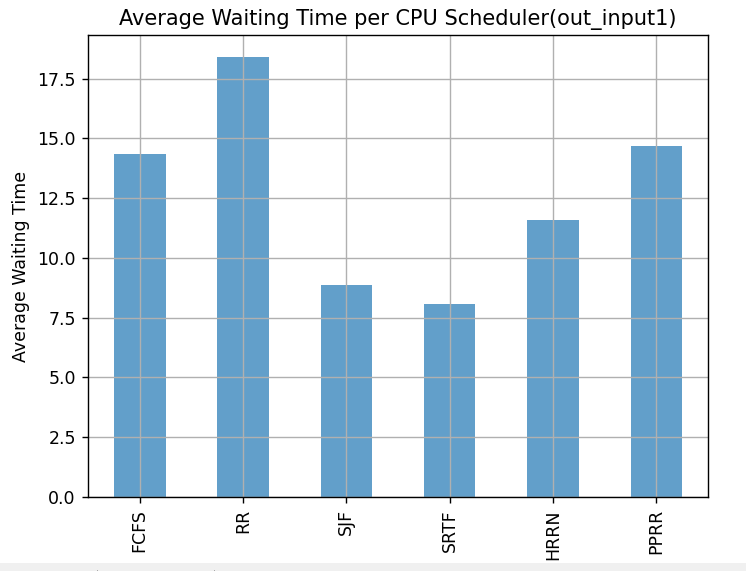
第四個SRTF跟上一個SJF幾乎相同，我這邊的只把之前的把用掉全部BURST改成每次就去只用1次(就是burst -1,clock+1)的意思，因為每次只跑一秒所以就可以處理要被奪取的狀況了，這樣持續到全部的CPU\_burst用完就結束了。

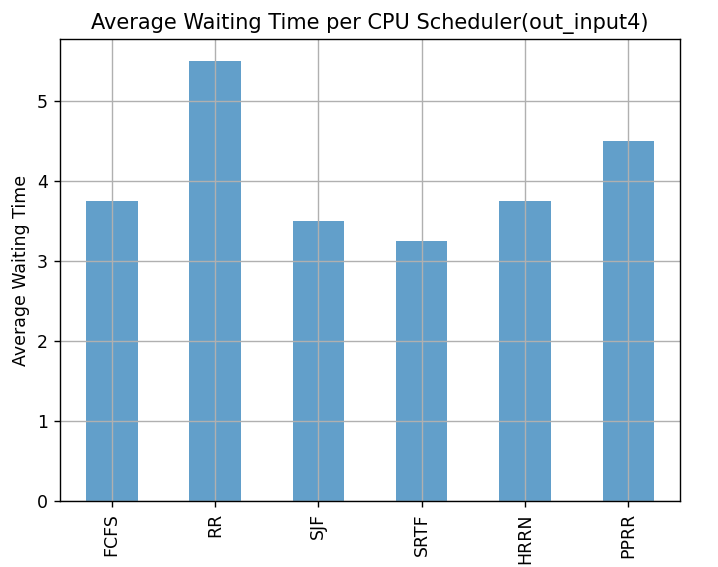
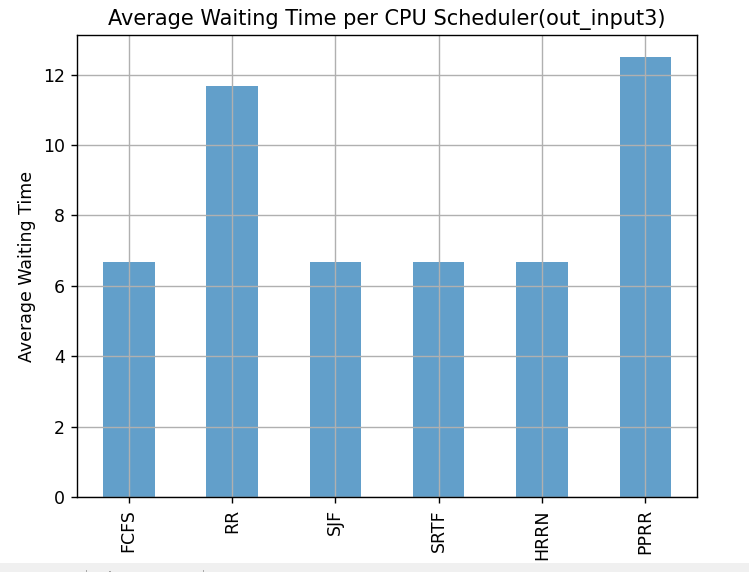
第五個HRRN排程，其實也跟SJF差不多，把最小的Burst改成了最高的Response rate，只要把queue裡面選出現在最高的Response rate(用現在clock和burst去算wt在去算Respose rate)，然後把這個擁有最高的Response rate的id給從queue中pop掉，然後做完他的Burst，這樣持續到全部用完就結束了。

第六個的PPRR比較麻煩一點，在把ID放入queue後要去依照Priority去排序，從排序好的拿第一項(最高priority的id)，在看下一項的Priority是不是跟他一樣，一樣的話就是要跑RR，沒有的話也是一樣只要跑一個timeslice，中途遇到的ID就把他加入到Queue遇到，並依照Priority排序後看第一項的Priority有沒有比現在的高(是否可奪取)，可奪取的話就重新append到queue裡面，用完timeslice後最後會看，如果用完了burst就繼續下一輪值到結束，如果沒用完burst，是被奪取或不是RR的話就不要從queue裡面pop掉，反之就先pop掉在重新append回去使其排到後面去。

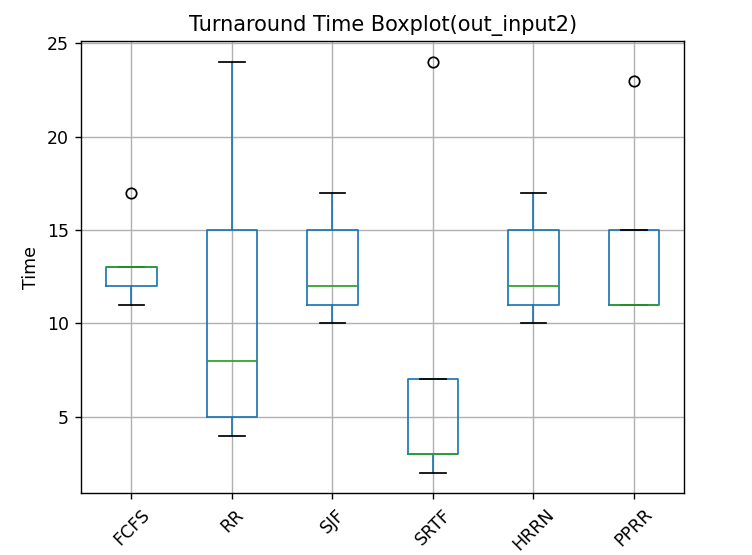
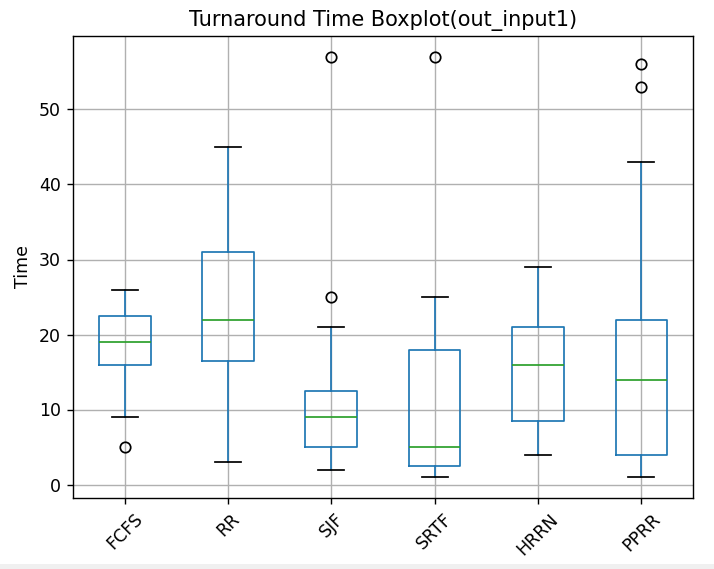
最後一個要實做的方法All只要把1~6的結果return整合成一分就好了。

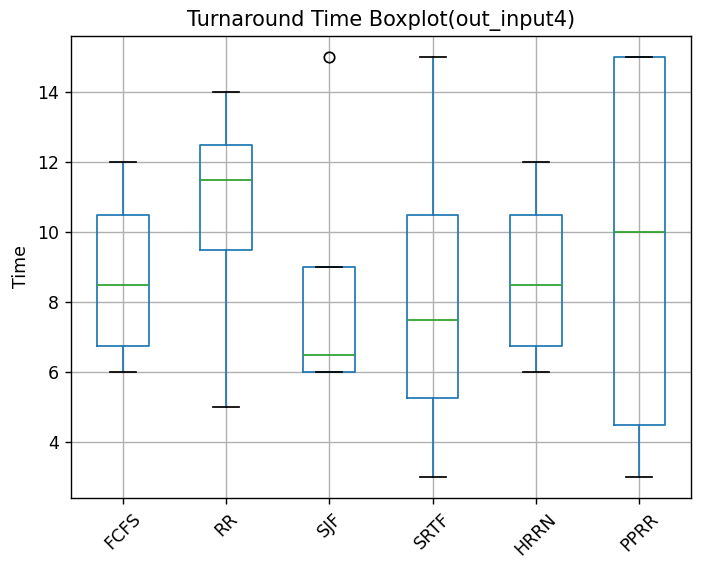
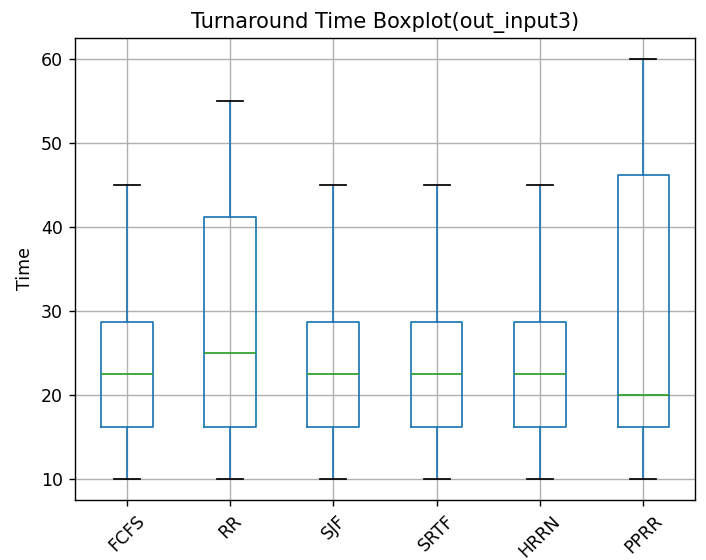
不同排程法的比較





上面是平均Waiting time的直方圖，對應到output到1~4，可以看出裡面最佳的排程法是SRTF在任何時刻都是最快的，而SJF次之，這是因為性質上最貪婪所以最快，但是現實是沒法精準預測CPU BURST的，看其他排程法中可看出HRRN在各排程法中效率站平均，看上去只比最普通的FCFS快一點點而已，而看得出RR和PPRR在選時間片段跟Priority的重要，output1的太小(1)，RR的效率甚至比FCFS還差，而output2選的剛好(3)，RR的效率明顯比其他的快，但是PPRR上就是因為Priority的問題，使著平均等待時間比其他都多(甚至比FCFS還多)。





Turnaround Time用盒狀圖可看出，FCFS,HRRN的盒子位置普遍(中位數和盒子等)偏平均，這代表著Turnaround Time在各方法中算篇平均的位置，而SJF和SRTF普遍(中位數和盒子等)偏下代表著Turnaround Time較低，RR和PPRR的偏高(中位數和盒子等)且盒子偏大，可能是因為Priority或timeslice的大小所影響，這代表著數據在上下四分位數之間相當的分散，但其實也很合理因為RR和PPRR這兩著就是用timeslice去處理，在大家都用timeslice下，大部分Turnaround time都會在同區間，至於在out\_input1可看到SJF和SRTF和PPRR有白點，這個是離異點就是在資料中跟大部分比起來算是異常值的資料，用SJF來想很好理解，因為小的Burst先處理，大的burst要到很後面才處理，故時間拉長就跟大部份資料差很多，看起來也會被視為異常值了。

結果與討論

這次的任務中從效能上來看，可得知用如果任務的CPU Burst time分布很平均的話，SJF和SRTF的表現會很明顯的好，因為他們可以優先去處理比較短的任務，FCFS和HRRN雖然在這次的任務的表現算平均，但是FCFS還是比較不適合的方法，因為先Arrival的是高CPU Burst的任務的話，那會使後面的任務被壅塞住，相較起來HRRN的排程法在這次全部的排成法中有著平衡的表現，因為每個任務都有機會，誰等得久或是比較小就比較優先，而RR和PPRR的話，就是要會抓timeslice的大小，抓得好的話可以使整體具有效率跟公平性，抓不好的效率甚至比FCFS還差，且PPRR的設計我覺得應該是把有先重要的任務附上高level一樣，感覺像是給電腦這種系統有高優先，其餘batch這種晚點再做一樣。

實際應用上，我覺得要看需求是怎麼樣如果對任務公平性較高的話，可以用RR或是HRRN去排，因為這兩者資源分配上較公平，不會使任務餓死，如果是反映時間的話，可以用SJF或SRTF去實做，不過這兩著的問題就是不好預測CPU BURST TIME的大小，如果是初學入門簡單測試的話，我覺得可以先去選FCFS當作基準，最後如果要運用到電腦這種複雜的架構的話，我覺得要用PPRR或者說是更進階版本(Multilevel feedback Queue)這類，讓系統這種重要的東西有著高優先度拿著多資源，並且作均衡的排程，PPRR雖然有缺點是當高優先度的任務接連而來時，會使低優先度的任務會餓死，沒有像Multilevel feedback queue可以隨著等待時間去升級(Aging)比較不重要的東西(batch)，把餓死的問題解決，但是Multilevel feedback queue的問題就是實作上困難，應用上PPRR也有好實作的優點就是了。