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。

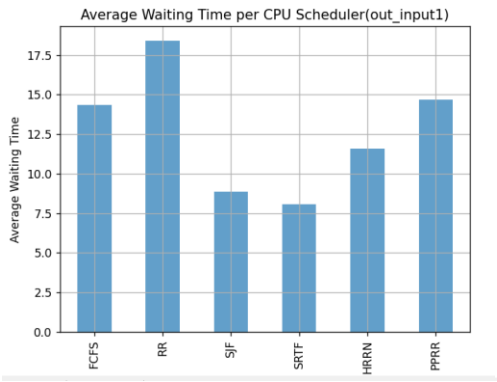
第四個 SRTF 跟上一個 SJF 幾乎相同，我這邊的只把之前的把用掉全部 BURST 改成每次就去只用 1 次(就是 burst -1,clock+1)的意思，因為每次只跑一秒所以就可以處理要被奪取的狀況了，這樣持續到全部的 CPU_burst 用完就結束了。

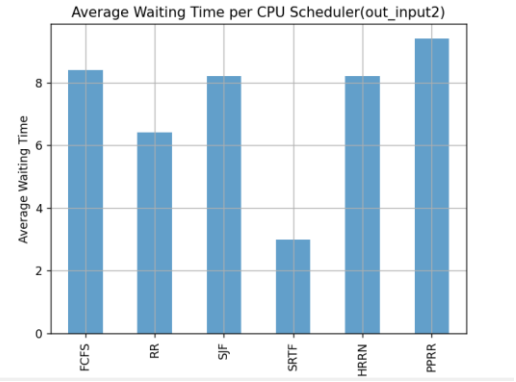
第五個 HRRN 排程，其實也跟 SJF 差不多，把最小的 Burst 改成了最高的 Response rate，只要把 queue 裡面選出現在最高的 Response rate(用現在 clock 和 burst 去算 wt 在去算 Respose rate)，然後把這個擁有最高的 Response rate 的 id 給從 queue 中 pop 掉，然後做完他的 Burst，這樣持續到全部用完就結束了。

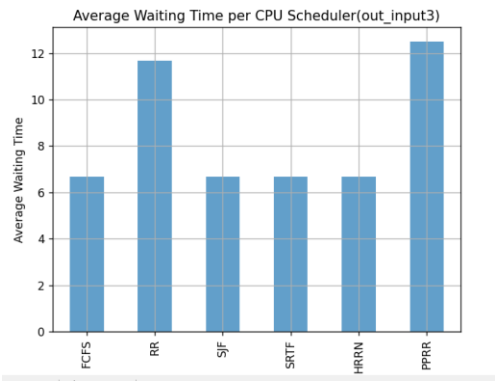
第六個的 PPRR 比較麻煩一點，在把 ID 放入 queue 後要去依照 Priority 去排序，從排序好的拿第一項(最高 priority 的 id)，在看下一項的 Priority 是不是跟他一樣，一樣的話就是要跑 RR，沒有的話也是一樣只要跑一個 timeslice，中途遇到的 ID 就把他加入到 Queue 遇到，並依照 Priority 排序後看第一項的 Priority 有沒有比現在的高(是否可奪取)，可奪取的話就重新 append 到 queue 裡面，用完 timeslice 後最後會看，如果用完了 burst 就繼續下一輪值到結束，如果沒用完 burst，是被奪取或不是 RR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

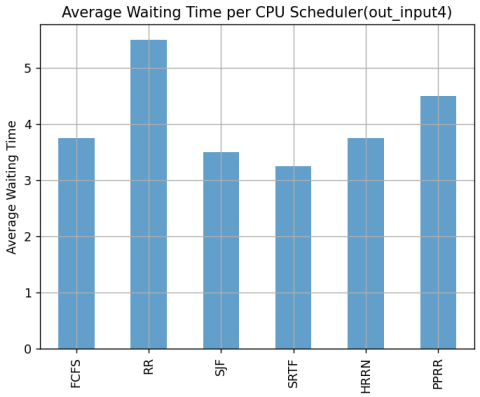
最後一個要實做的方法 All 只要把 1~6 的結果 return 整合成一分就好了。

不同排程法的比較

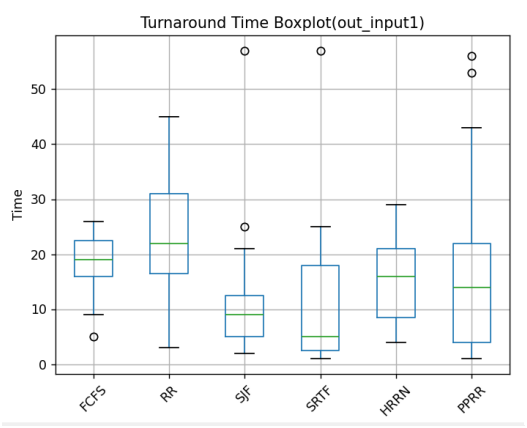


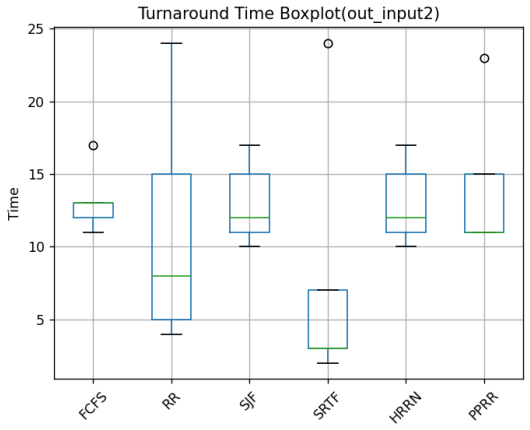


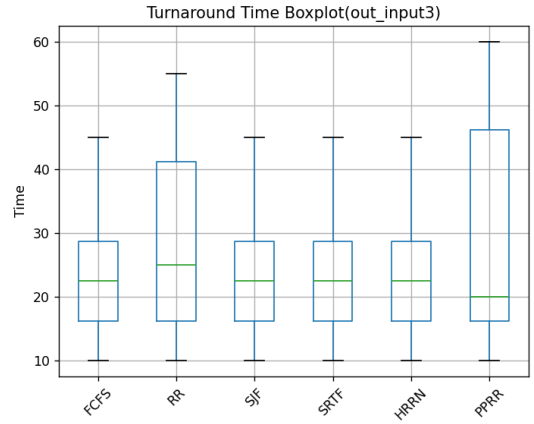


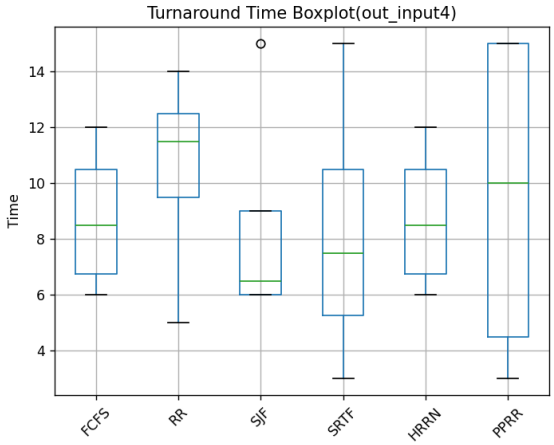


上面是平均 Waiting time 的直方圖，對應到 output 到 1~4，可以看出裡面最佳的排程法是 SRTF 在任何時刻都是最快的，而 SJF 次之，這是因為性質上最貪婪所以最快，但是現實是沒法精準預測 CPU BURST 的，看其他排程法中可看出 HRRN 在各排程法中效率站平均，看上去只比最普通的 FCFS 快一點點而已，而看得出 RR 和 PPRR 在選時間片段跟 Priority 的重要，output1 的太小(1)，RR 的效率甚至比 FCFS 還差，而 output2 選的剛好(3)，RR 的效率明顯比其他的快，但是 PPRR 上就是因為 Priority 的問題，使著平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