1. 開發環境

Windows11,Anaconda Python 3.8(需安裝程式裡面含有的package才能正常的執行)

2. 實作方法和流程

FCFS:利用python dictionary儲存資訊，對arrival time, ID進行排序，而就照排好的序列一個一個做完，當其他程序再等待就計算每個waiting time，做完就設置此程序的turnaround time。

RR: 利用python dictionary儲存資訊，對arrival time, ID進行排序，利用queue把在current time時間內的排序好的程序一個一個加進queue，加完之後，再pop一個出來做一個time slice如果做完或time out先把小於current time的程序加入queue，再把沒做完的程序加回queue，假如做完就不用加回queue裡。當其他程序再等待就計算每個waiting time，做完就設置此程序的turnaround time。

SRTF:按照CPU Burst, Arrival Time, ID排序，每做一個時間單位就按照CPU Burst, Arrival Time, ID排序且更新一次，反覆直到做完，當其他程序再等待就計算每個waiting time，做完就設置此程序的turnaround time。

PPRR: 按照Priority, Arrival Time, ID排序，也使用了queue，最前面的一定是最高優先且先做的程序，每做一個時間單位就更新一次看有無更高Priority的程序，當其他程序再等待就計算每個waiting time，做完就設置此程序的turnaround time。

HRRN:用一個list，在做完一個程序就更新一次在時間內可執行程序的Response ratio，按照這個大小排序，拿最大的程序出來做完，在重複更新一次，重複上述步驟直到做完，當其他程序再等待就計算每個waiting time，做完就設置此程序的turnaround time。

3. 不同排程法的比較

FCFS和RR基本上演算法差不多，就我在寫這兩個排程的時候，發現到兩個的差異是RR比較公平一點，但就只是加了time slice的FCFS，假如time slice太大基本上排程的結果就會和FCFS一樣，time slice太小的話就是一直頻繁的切換process其實還蠻浪費效能和效率的。

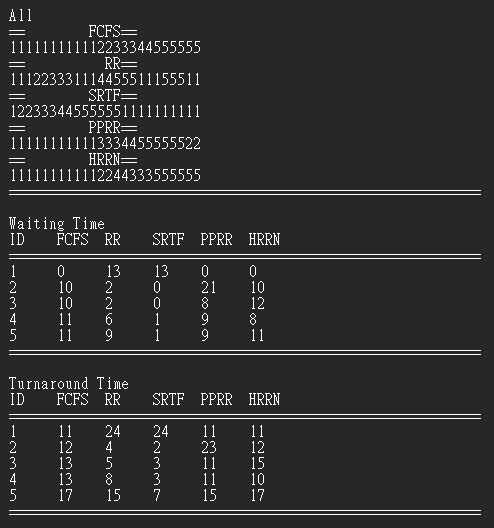
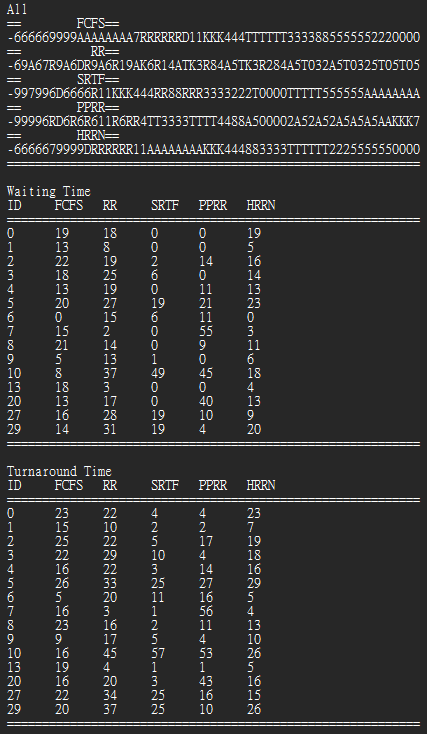
SRTF可以減少平均的等待時間，原因就是當還沒完成工作的程序越多，那等待的程序就越多，平均等待時間就越大，所以這個排程先把快結束的程序處理，優點是可以減少等待程序的數量進而減少平均等待時間且注重執行時間，缺點是沒有注意等待的時間。

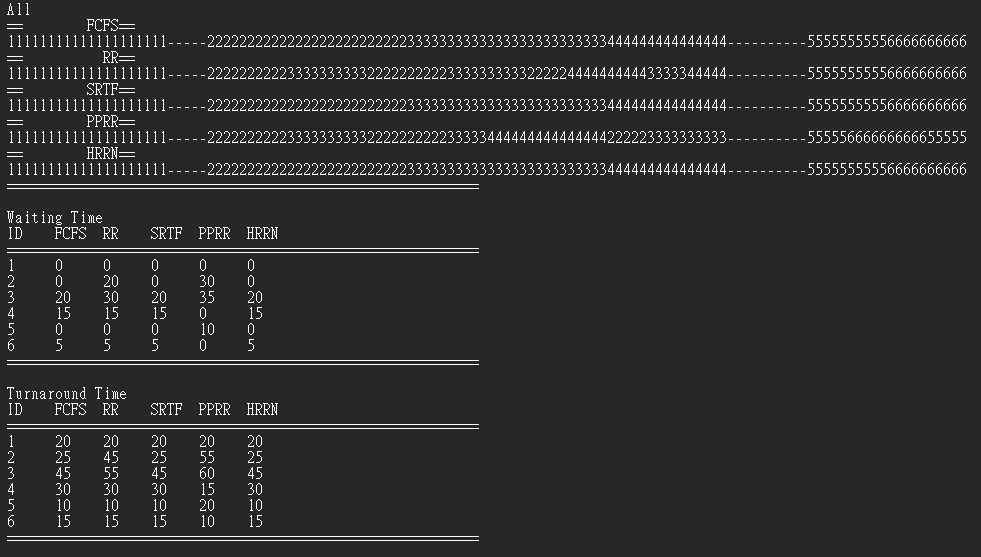
PPRR就是加上Priority的RR排程，只考慮Priority，當前High Priority的排程會最先做完，除非有更高優先權的程序進來，但假設一直有更高優先度的程序一直加進來會導致很早來但優先度低的程序會使turnaround time越來越大，而程序有可能會餓死，在這次作業並沒有加入時間升級的機制，所以此排程是有可能讓程序餓死的狀況。

HRRN

注重執行的時間和等待的時間，優先順序會因時間的增加，使等待的程序的優先程度上升，一開始比例最高的會先做完，而做完之後算比例，再挑下一個比例高的做完，反覆地直到沒有下一個程序。

4. 結果與討論





SRTF真的減少了平均等待時間，對於cpu burst低的turnaround time也變小，time slice很小對於RR並沒有太大的幫助，必須一直context switch，假設程序能在time slice裡做完會比較接近FCFS，PPRR算是針對RR做一個優先權的排序，高優先度的會比較先做完，HRRN的結果有點類似FCFS或是RR，都是多出一種INDEX做排序的機制，HRRN多出了Response Ratio的排序機制，也就是High response ratio first，比較可以將waiting 比較久時間的程序把waiting時間分布給那些沒有等很久的程序。