作業系統
HW2-書面報告
班級:資訊三甲
學號:**11027142**
姓名:林蕙郁

- 開發環境
  利用Python程式碼以及Python 3.11.4('base':conda)的Interpreter實作。

- 實作方法和流程

  讀入檔案後，先讀取第一行的method number跟time slice，將下來我會將資料內容(PID, CPU_burst, arrival_time, priority)利用append放入一個叫做processes的list內，最終做出一個擁有各項資料的list。

  FCFS:我先利用processes.sort(key=sort_key_arr_pid)去做排序，其中key後所接的方法是另外寫了一個def，判斷標準是先看arrival_time，如果arrival_time相同的話就去看PID，最終的數據會是從小排到大。
  排完之後去processes看arrival_time跟current_time的關係，如果不符合條件(processes['arrival_time']>current_time)，就直接利用將current_time賦值成processes[0]['arrival_time']且計算idle time並加入到甘特圖中。接下來因為確定了current_time，先把它紀錄下來，然後將執行時間加上cpu_burst變成completion_time，同時也要記錄甘特圖。最後就是去計算waiting_time跟turnaround_time，做完之後就繼續看processes還有沒有沒執行到的process，如果有的話就繼續執行直到全部process都做完，最終return 甘特圖、waiting_time_list跟turnaround_time_list去寫檔。

  RR:和FCFS一開始進去一樣先做排序，在這一題我是利用deque實作。如果processes裡面有process且他的arrival_time<=current_time，就加入到queue裡面。接下來看queue是不是空的，不是的話我就要將queue的第一個資料拿出來，判斷他的cpu burst跟time slice的大小。如果time slice比較小的話，代表他最後會因為timeout而重新排隊，反之，他會將這個process做完並且離開queue。確定好他的execute time之後我就會去更新current time跟增加新的PID進入甘特圖。waiting time是每次只要又執行到同一個process的時候要把時間累加上去，而turnaround time則是我確定我的cpu burst==0之後再去做計算。同樣的事情會一直持續到processes內沒有任何process需要被執行，最終return要寫檔案的資料回去(甘特圖、waiting time list、turnaround time list)。

  SJF:先對processes做排序(cpu burst->arrival_time->PID)。只要current time跟process的arrival time符合條件，我就會將該process加入要被執行的隊伍裡面(也就是進去排隊)，如果沒有的話(也就是剛好是idle time)，我就會直接去找最近的process加進去要被執行的隊伍中，接下來去取出佇列中cpu burst最小的做排程，會記錄甘特圖、更新當下時間、waiting time、turnaround time，然後把這個process移出processes，最後迴圈去檢查processes還有沒有process沒有被執行到，如果都執行完畢的話就結束。(同樣return要寫檔的資料)

  SRTF:先對processes做排序(arrival_time->cpu burst->PID)，當arrival time跟current time符合條件，我利用heapq去insert到ready_queue(會看cpu_burst、arrival_time、pid去決定在插入在哪裡)，接下來會去ready_queue裡面找cpu curst最小的作執行，直到執行到下一個processes裡的process進入到ready queue，記錄執行時間、更新current time跟甘特圖、更新current time還有剩餘

cpu burst還有多少，如果cpu burst>0就要重新排隊，如果==0就計算turnaround time跟waiting time。最終把所有processes內的process做完之後return要寫入檔案的資料。

HRRN：如果arrival time <= current time就要判斷他的ratio，如果ratio>highest ratio的話，就要更新最高比率的數字以及將ratio_list更改為目前這個ratio的process內容，如果算出來的比率跟最高比率相同，就把這個process append到這個highest ratio list內。如果highest ratio list不是空的話，就去對他的arrival time跟PID做排序，且取出要執行的process去計算執行時間、更新甘特圖、更新current time、waiting time、turnaround time。如果是空的就去找最近的process並將idle time的內容加入甘特圖中。最終把所有processes做完之後就return要寫入檔案的資料。

PPRR：先對processes做排序（依照arrival_time->priority->pid），然後去判斷arrival_time跟current time的關係，如果相同的話我會把他加入一個samearr_list（為了符合第四點，如果有不只一個process進入的話，要依照prioirty後再排序PID去做插入），如果只有一個就直接加入到佇列尾端就好，其他arrival_time<current_time的就依照priority去排序就好，而儲存這些資料的資料型別我是利用dict，儲存樣子會像{priority:pid, arrival_time, cpu_burst, priority}。然後在這題我是採用執行一秒就去判斷會不會被搶奪、逾時或是做完process，每次開始我都會去找優先度最大的process先做，且會記錄這一秒的process來為下一秒做準備（判斷是否搶奪），而timeout的部分我採用看timeslice還剩多少，如果==0的話就是timeout，我會讓他重新排隊並找下一個process開始做，如果是cpu_burst變成0的話我就會更新等待時間跟turnaround time。每秒也都會更新甘特圖。直到把所有processes做完，return要寫入檔案的資料。

- 不同排程法的比較

下表為所有檔案的所有排程法之平均等待時間

平均等待時間	FCFS	RR	SJF	SRTF	HRRN	PPRR
input1	14.333	18.4	8.87	8.06	11.6	14.67
input2	8.4	6.4	8.2	5	8.2	9.4
input3	6.67	11.6	6.67	6.67	6.67	12.5
input4	3.75	5.5	3.5	3.25	3.75	4.5

由此表觀察出以下幾點：
1：FCFS的等待時間取決於process arrival time跟cpu burst，如果很幸運在結束一個process的時候剛好下一個process剛到，或是cpu burst越小的話，則等待時間就會很短，反之，如果抵達時間相近，cpu burst越大，則等待時間就會一直延長。

2:同個檔案在做RR跟PPRR的時候消耗時間都會較久，因為可能會因為timeout導致他需要重新排隊一次，且這個排隊就會導致他的waiting time上升（也可以說又累加上去），所以最終的平均當然會比較高。
而在這裡input1的特別久是因為他的time slice很小（數值為1），所以很容易會有timeout的情況出現，如果time slice切大一點就可以避免頻繁過度timeout+重新排隊的狀況。
3:SRTF算是等待時間最短的一個排程法，因為幾乎能一直執行process，且只要發現有cpu burst更小的他就會優先執行，不會像RR有需要一直重新排隊的問題，也不會像FCFS必須等到當前process做完才有辦法做下一個process。

下表為平均turnaround time

平均 turnaround	FCFS	RR	SJF	SRTF	HRRN	PPRR
input1	18.2	22.26	12.73	11.93	15.47	18.53
input2	13.2	11.2	13	7.8	13	14.2
input3	24.16	29.16	24.16	24.16	24.16	30
input4	8.75	10.5	8.5	8.25	8.75	9.5

下表針對time slice不同，RR排程去做比較。

time slice(RR) & waiting time	1	10	50
input2	6.6	10.2	8.4
input3	13.16	11.66	6.67

說明:
如果time slice很大的話，效率會像FCFS一樣，幾乎不太會有timeout的發生，所以算是一個process可以佔據很長的時間，甚至有可能可以直接讓他做完。
如果time slice很小的話就會導致他常常timeout，重新排隊，等待時間就會繼續加長，最後的平均數值也會比較大。

- 結果與討論

  根據上面六個排程法，我提供一些我認為適合的場景:

1.FCFS：適用於arrival time不相近，CPU burst較小時表現會較好。

2.RR：time slice適當的話每個人得到的時間都是公平的，可以輪流使用資源是很優秀的優點，但是time slice切太大切大小仍可能會導致出現缺點。
切太小：一直timeout+重新排隊，等待時間大幅提升。
切太大：會如同FCFS的概念。

3.：SJF：CPU burst不能太大（不然可能會導致餓死），優點是可以讓process完整的執行完，但因為不能預先知道下一個process的CPU burst，所以能應用場景並不像SRTF那麼多。

4.SRTF：算是表現上最好的，因為可以一直讓process被執行，不太會有idle time，等待時間也會是最短的，但CPU不一定永遠能事先知道所有process的CPU burst time，所以實務上還有點難達成，但以理論來說他算是非常優秀的排程法。

5.HRRN：增加了Aging防止餓死，但每次有新的process進入排隊佇列的時候需要重新計算所有的response ratio，如果資料量很大的話可能會耗費大量資源跟時間。

6.PPRR：藉由優先度決定誰先做，也增加RR防止一個人占用資源太久（time slice切的合適的情況下），讓大家可以輪流使用，有雙重保障的概念。