Assignment 8 –Multilevel Queue Scheduling

Problem given and solution I made

Question was to write a program for Multilevel Queue Scheduling algorithm by considering for queues. I programed a multilevel queue in language Python without considering arrival time of each processes. All the processes arrives at current_time 0.

In this multilevel queue there are four queues

1) Q0 – Round Robin(RR)
2) Q1-Shortest Job First(SJF) 1
3) Q2-Shortest Job First(SJF) 2
4) Q3-First Come First Serve(FCFS/FIFO)

Round Robin Queue is a Circuar Queue which keep processes and which shares time with the processes in that queue. These process share time with a quantum time. Shortest Job first queues are the queues which enable to take CPU for the process of minimum Burst Time in order. First Come First Serve are the general CPU scheduling queues which takes processes which arrives to the ready queues in order.

In this multilevel queue 20 sec of switching time is enabled. Processes at Q0 executes for 20 seconds, Processes at q1 executes for 20 seconds , Processes at q2 executes for 20 seconds, Processes at q3 executes for 20 seconds and then again q0, so it repeats until all the queues become emptied. To execute the process in these queues they should become non emptied

Implementation

- In my program I created a python class named Process which contains attributes of process id (pid), burst time(burst_time), priority, remaining time (remaining_time), waiting time (waiting_time) ,turnaround time (turnaround_time).

```
class Process:
    def __init__(self, pid, bt, p):
        self.pid = pid
        self.burst_time = bt
        self.priority = p
        self.remaining_time = bt
        self.waiting_time=0
        self.turnaround_time=0
```

I've created some global integer variables which can be accessed even by all the functions I created . These variables are based on time related operations we do in the program.

Timer is for switching the queues by 20 seconds. Timer is reduced due to the occasions and it again resets to 20 when another queues starts to execute its processes

```
timer = 20
current_time=0
current_queue = 0
time_Quantem=5

total_waiting_time_for_Roundrobin=0
total_turnAround_time_for_Roundrobin=0
total_waiting_time_for_SJF1=0
total_waiting_time_for_SJF2=0
total_turnAround_time_for_SJF1=0
total_turnAround_time_for_SJF2=0
total_waiting_time_for_FCFS=0
total_turnAround_time_for_FCFS=0
```

Current_time is the global variable which increases due to the time according to the operations.

Current_queue is the recently executed queue. When switching to the next queues we check , where CPU was. And when we completed a process in a queue completely we check the current_queue to execute next process in that queue.

Time_Quantem is used for share time between processes in RoundRobin Queue. I took it as 5 seconds

Total waiting times and turnaround times for each queues are used to find total and average times for waiting and turnaround Times

```python
q=[[],[],[],[]]
n0=0
n1=0
n2=0
n3=0
n = int(input("Enter the number of processes: "))
print("Enter the priority, burst time for each process: \n--------
for i in range(n):
    print("Enter details of process " + str(i+1))
    print("Enter burstTime : ");
    burstTime=int(input())
    print("Enter priority : ");
    priority = int(input())
    print("\n")
    if priority == 0:
        q[0].append(Process((i+1),burstTime, priority))
        n0+=1
    if priority == 1:
        q[1].append(Process((i+1), burstTime, priority))
        n1+=1
    if priority == 2:
        q[2].append(Process((i+1), burstTime, priority))
        n2+=1
    if priority == 3:
        q[3].append(Process((i+1), burstTime, priority))
        n3+=1
q[1].sort(key = lambda process: process.burst_time)
q[2].sort(key = lambda process: process.burst_time)
```

q is the 2D List which contains the 4 queues. Integer variables n0,n1,n2,n3 are created and initialized to 0. Number of processes we insert is entered by the user.

Then the for loop works. It input details of each processes and enqueued to each queues according to the priority we insert.

As q[1] and q[2] are shortest job first queues. I sort those two lists according to the burst times.

The burst time we inserted is inserted to each processs by the constructor we created before. Remaining time also contains the burst time inserted initially.

dequeueRoundRobin() Function

```python
def dequeueRoundRobin():
    global timer
    global current_time
    global  total_turnAround_time_for_Roundrobin
    global total_waiting_time_for_Roundrobin
    if timer >= time_Quantem:
        p = q[0][0]
        print("Process id : "+str(p.pid)+" takes CPU to execute now at "+str(current_time))
        if p.remaining_time > time_Quantem:
            timer -= time_Quantem
            p.remaining_time -= time_Quantem
            current_time+=time_Quantem
            if timer == time_Quantem:
                print("The process with process id " + str(p.pid) + " completed successfully at "+str(current_time))
                p.turnaround_time=current_time
                total_turnAround_time_for_Roundrobin+=p.turnaround_time
                p.waiting_time=p.turnaround_time-p.burst_time
                total_waiting_time_for_Roundrobin+=p.waiting_time
                print("turndaround time for "+str(p.pid)+" is : "+str(p.turnaround_time))
                print("waiting time for " + str(p.pid) + " is : " + str(p.waiting_time)+"\n")
                q[0].pop(0)
            else:
                q[0].append(p)
                q[0].pop(0)
        elif p.remaining_time == time_Quantem:
            timer -= time_Quantem
            p.remaining_time -= time_Quantem
            current_time += time_Quantem
            print("The process with process id " + str(p.pid) + " completed successfully at " + str(current_time))
```

Dequeueroundrobin() function is created here. Global variables timer, current_timer and total turnaround_time and waiting_time for roundrobin are used here. First we check whether switching time is higher or equal to the time quantem. If so we take front process at the round robin queue as the process p.

Then we check remaining_time >= time quantum to check whether process can not be completely executed within a time quantum. If process can not be completely executed within a time quantum. Process run for the quantum. So timer decreased by time_quantem. Remaining time reduces by time_quantem. Current_time increases by time_quantem. Then if timer==time_quantem process completes on time and popped. If timer>time_quantem process can not be completed. It executes for quantum_time. And rest of the process is enqueued to the queue.

Then remaining_time==timeQuantem condition filter out the processes with remaining time equals to time quantum then the timer is reduced, remaining time is reduced and current time is increased by time quantum of time. Then turnaround times and waiting time have been calculated and the completed process was poped from the queue.

```
            q[0].pop(0)
    else:
        timer -= p.remaining_time
        current_time+=p.remaining_time
        p.remaining_time = 0
        print("The process with process id " + str(p.pid) + " completed successfully at "+str(current_time))
        p.turnaround_time = current_time
        total_turnAround_time_for_Roundrobin += p.turnaround_time
        p.waiting_time = p.turnaround_time - p.burst_time
        total_waiting_time_for_Roundrobin += p.waiting_time
        print("turndaround time for " + str(p.pid) + " is : " + str(p.turnaround_time))
        print("waiting time for " + str(p.pid) + " is : " + str(p.waiting_time)+"\n")
        q[0].pop(0)
else:
    p = q[0][0]
    print("Process id : "+str(p.pid)+" takes CPU to execute now at "+str(current_time))
    if p.remaining_time <= timer:
        timer -= p.remaining_time
        current_time+=p.remaining_time
        p.remaining_time = 0
        print("The process with process id " + str(p.pid) + " completed successfully at "+str(current_time))
        p.turnaround_time = current_time
        total_turnAround_time_for_Roundrobin += p.turnaround_time
        p.waiting_time = p.turnaround_time - p.burst_time
        total_waiting_time_for_Roundrobin += p.waiting_time
        print("turndaround time for " + str(p.pid) + " is : " + str(p.turnaround_time))
        print("waiting time for " + str(p.pid) + " is : " + str(p.waiting_time)+"\n")
        q[0].pop(0)
    else:
        p.remaining_time -= timer
```

Then else part is for the processes which can be completed before time quantum of time. If so front process is taken in to process p. then remaining time is enough to be executed before switching to the next queue is checked. If true timer reduces , current time increases, by remaining time of the process.

After completing the process, the completed process is poped from the queue. Then the condition timer<time_quantem is checked. Front process is taken as the process p. then if remaining time <= timer is true it means process can be completed before switching to the next queue. Timer is reduced, current_time is increased by remaining time. Process is completed. If remaining time >timer remaining time is decreased, current time is increased by timer of time. Then CPU is ready to switch to the next unemptied queue.

The turnaround times and waiting time are calculated due to there definitions. Turnaround time is how much time consumed for complete the process. Waiting time is how much time process waits at the queue to be executed.

Turnaround time can be calculated by taking the current_time when it completed executing.

As all processes arrived at time 0. We don't consider the arrival time simply take the current_time. Waiting time can be calculated by taking turnaround time – burst time. Total values can be taken when adding those values to the global variable total_turnAround_time_for_roundrobin, and total_waiting_time_for_roundrobin.

dequeueSjf() function

```python
def dequeueSjf(sjfQueue):
    global timer,current_time
    global total_turnAround_time_for_SJF1,total_turnAround_time_for_SJF2,total_waiting_time_for_SJF1,total_waiting_time_for_SJF2
    p = sjfQueue[0]
    print("Process id : "+str(p.pid)+" takes CPU to execute now at "+str(current_time))
    if p.remaining_time <= timer:
        timer -= p.remaining_time
        current_time+=p.remaining_time
        p.remaining_time = 0
        sjfQueue.pop(0)
        print("The process with process id " + str(p.pid) + " completed successfully at "+str(current_time))
        p.turnaround_time = current_time
        p.waiting_time = p.turnaround_time - p.burst_time
        if sjfQueue==q[1]:
            total_turnAround_time_for_SJF1 += p.turnaround_time
            total_waiting_time_for_SJF1 += p.waiting_time
        elif sjfQueue==q[2]:
            total_turnAround_time_for_SJF2 += p.turnaround_time
            total_waiting_time_for_SJF2 += p.waiting_time

        print("turndaround time for " + str(p.pid) + " is : " + str(p.turnaround_time))
        print("waiting time for " + str(p.pid) + " is : " + str(p.waiting_time)+"\n")
    else:
        p.remaining_time -= timer
        current_time+=timer
        timer = 0
```

dequeueSjf() function and dequeueFCFS() functions are simpler than the roundrobin function.

Two shortest job first queues are passed in to the paramteters. So this function is for those two queues. Global variable. Global variables are used here to access and change them according to the exexcution. First we take front process to the process p. CPU starts to use that process. If remaining time<= timer means if we can switch execute the process p before switch to the next queue. Then current time is increased , timer is decreased by remaining time of process p. then process successfully completes its execution and calculates turnaround time and waiting time.

If remaining time is higher than the timer means, process cannot be competed before switching to the next queue. Then remaining time is reduced , current time is increased by the timer.

dequeueFcfs() function

```
def dequeueFcfs():
    global timer,current_time
    global total_turnAround_time_for_FCFS,total_waiting_time_for_FCFS
    p = q[3][0]
    print("Process id : "+str(p.pid)+" takes CPU to execute now at "+str(current_time))
    if p.remaining_time <= timer:
        timer -= p.remaining_time
        current_time+=p.remaining_time
        p.remaining_time = 0
        q[3].pop(0)
        print("The process with process id " + str(p.pid) + " completed successfully at "+str(current_time))
        p.turnaround_time = current_time
        total_turnAround_time_for_FCFS += p.turnaround_time
        p.waiting_time = p.turnaround_time - p.burst_time
        total_waiting_time_for_FCFS += p.waiting_time
        print("turndaround time for " + str(p.pid) + " is : " + str(p.turnaround_time))
        print("waiting time for " + str(p.pid) + " is : " + str(p.waiting_time)+"\n")
    else:
        p.remaining_time -= timer
        current_time+=timer
        timer = 0
```

This refer to Dequeue function for fcfs. Global integer variables timer, current_time and total of turnaround time and waiting time is accessed and changed here. First of all front process is taken as the process p. then functions starts.

Remainingtime<=timer means when time before switch to the next queue is enough for process p's remaining time. If so tier is reduced, current_time is increased by remaining time. Then turnaround time and waiting time are calculated.

Else part refers to the condition remaining_time>timer. This means when time remains for the process is higher than the time left to switch to the next queue. CPU from this queue to next queue before complete the process. So remaining time is decreased, current_time is increased by the timer.

So the three dequeue functions are over the next function is the condition () function. For 4 queues there are special functions. I am using a loop to dequeue functions later by considering current queue and non empty queues left. So there should be condition function which can be called to do the same activity again by again by changing the parameters.

After calling each functions the status of the queue is saved in current_queue variable.

```
def condition(k):
    global current_queue
    if k==0:
        dequeueRoundRobin()
        current_queue = 0
    elif k==1:
        dequeueSjf(q[1])
        current_queue = 1
    elif k==2:
        dequeueSjf(q[2])
        current_queue = 2
    elif k==3:
        dequeueFcfs()
        current_queue = 3
```

```
j=0
while(len(q[0]) != 0) or (len(q[1]) != 0) or (len(q[2]) != 0) or (len(q[3]) != 0):
    if timer == 0:
        timer = 20
        if (current_queue == j):
            if (len(q[(j + 1)%4])!=0):
                condition((j+1)%4)
            elif(len(q[(j + 2)%4])!=0):
                condition((j + 2) % 4)
            elif (len(q[(j + 3) % 4]) !=0):
                condition((j + 3) % 4)
            elif (len(q[(j + 4) % 4]) !=0):
                condition((j + 4) % 4)
    else:
        if (current_queue == j):
            if (len(q[(j) % 4]) != 0):
                    condition((j) % 4)
            elif (len(q[(j + 1) % 4]) != 0):
                    condition((j + 1) % 4)
            elif (len(q[(j + 2) % 4]) != 0):
                    condition((j + 2) % 4)
            elif (len(q[(j + 3) % 4]) != 0):
                    condition((j + 3) % 4)
        j = (j + 1) % 4
```

While refers to the occasions when there are one or more process at any queue. If we find no process at any queue the loop ends. Then checked whether timer==0 or not. This means the occasion when cpu is to switch to the next avalible queue. If the cpu is in switching point cpu checks the current queue and next avalible non empty queue which contains processes. And calls the condition () function. Arguments are passed to find the next avalible queue with processes. On that condition () function calls to the related dequeue method according to the queues.

If timer!=0 it means there is some time to execute a process at the current queue. So it checks the current queue. Then it is checked the current queue and the next queues are empty or not. The first non emptied queue takes cpu's attention.

There are only for queues 0 to 3 because of that I used j = (j+1)%4 to change variable to move between queues.

Then after completing successfully the process. All the total turnaround times and waiting times are displayed.

```
print("Processes successfully executed............................................\n")
print("Total waiting time and turn around time calculated\n")

print("Total waiting time for roundrobin : "+str(total_waiting_time_for_Roundrobin))
print("Total TurnAround time for roundrobin : "+str(total_turnAround_time_for_Roundrobin))
print("Total waiting time for SJF1 : "+str(total_waiting_time_for_SJF1))
print("Total TurnArond time for SJF1 : "+str(total_turnAround_time_for_SJF1))
print("Total waiting time for SJF2 : "+str(total_waiting_time_for_SJF2))
print("Total TurnArond time for SJF2 : "+str(total_turnAround_time_for_SJF2))
print("Total waiting time for FCFS : "+str(total_waiting_time_for_FCFS))
print("Total TurnArond time for FCFS : "+str(total_turnAround_time_for_FCFS))

print("\nAverage waiting time and turn around time calculated\n")
print("Average waiting time for roundrobin : "+str(total_waiting_time_for_Roundrobin/n0))
print("Average TurnAround time for roundrobin : "+str(total_turnAround_time_for_Roundrobin/n0))
print("Average waiting time for SJF1 : "+str(total_waiting_time_for_SJF1/n1))
print("Average TurnArond time for SJF1 : "+str(total_turnAround_time_for_SJF1/n1))
print("Average waiting time for SJF2 : "+str(total_waiting_time_for_SJF2/n2))
print("Average TurnArond time for SJF2 : "+str(total_turnAround_time_for_SJF2/n2))
print("Average waiting time for FCFS : "+str(total_waiting_time_for_FCFS/n3))
print("Average TurnArond time for FCFS : "+str(total_turnAround_time_for_FCFS/n3))
```

Testing

Test case 1

Processes with these burst_times and priorities are arrived to the queues at the current_time=0

Process ID	Burst Time	Priority
1	10	0
2	3	0
3	4	0
4	4	1
5	2	1
6	1	1
7	5	2
8	3	2
9	2	2
10	4	3
11	2	3
12	6	3

All theses processes are inserted to the queues at the same arrival_time of 0 sec. Then the queues will be like this.

Front		**Rear**	
10 p1	3 p2	4 p3	q0
1 p6	2 p5	4 p4	Q1
2 p9	3 p8	5 p7	Q2
4 p10	2 p11	6 p12	

Process should be completed like this.

P2 ,P3, P1 ,P6, P5 ,P4, P9, P8, P7, P10, P11, P12

```
Process id : 1 takes CPU to execute now at 0
Process id : 2 takes CPU to execute now at 5
The process with process id 2 completed successfully at 8
turndaround time for 2 is : 8
waiting time for 2 is : 5

Process id : 3 takes CPU to execute now at 8
The process with process id 3 completed successfully at 12
turndaround time for 3 is : 12
waiting time for 3 is : 8

Process id : 1 takes CPU to execute now at 12
The process with process id 1 completed successfully at 17
turndaround time for 1 is : 17
waiting time for 1 is : 7

Process id : 6 takes CPU to execute now at 17
The process with process id 6 completed successfully at 18
turndaround time for 6 is : 18
waiting time for 6 is : 17

Process id : 5 takes CPU to execute now at 18
The process with process id 5 completed successfully at 20
turndaround time for 5 is : 20
waiting time for 5 is : 18

Process id : 4 takes CPU to execute now at 20
The process with process id 4 completed successfully at 24
turndaround time for 4 is : 24
```

```
Process id : 9 takes CPU to execute now at 24
The process with process id 9 completed successfully at 26
turndaround time for 9 is : 26
waiting time for 9 is : 24

Process id : 8 takes CPU to execute now at 26
The process with process id 8 completed successfully at 29
turndaround time for 8 is : 29
waiting time for 8 is : 26

Process id : 7 takes CPU to execute now at 29
The process with process id 7 completed successfully at 34
turndaround time for 7 is : 34
waiting time for 7 is : 29

Process id : 10 takes CPU to execute now at 34
The process with process id 10 completed successfully at 38
turndaround time for 10 is : 38
waiting time for 10 is : 34

Process id : 11 takes CPU to execute now at 38
The process with process id 11 completed successfully at 40
turndaround time for 11 is : 40
waiting time for 11 is : 38

Process id : 12 takes CPU to execute now at 40
The process with process id 12 completed successfully at 46
turndaround time for 12 is : 46
waiting time for 12 is : 40
```

```
Processes successfully executed............................

Total waiting time and turn around time calculated

Total waiting time for roundrobin : 20
Total TurnAround time for roundrobin : 37
Total waiting time for SJF1 : 84
Total TurnArond time for SJF1 : 96
Total waiting time for SJF2 : 50
Total TurnArond time for SJF2 : 55
Total waiting time for FCFS : 112
Total TurnArond time for FCFS : 124

Average waiting time and turn around time calculated

Average waiting time for roundrobin : 6.666666666666667
Average TurnAround time for roundrobin : 12.333333333333334
Average waiting time for SJF1 : 28.0
Average TurnArond time for SJF1 : 32.0
Average waiting time for SJF2 : 16.666666666666668
Average TurnArond time for SJF2 : 18.333333333333332
Average waiting time for FCFS : 37.333333333333336
Average TurnArond time for FCFS : 41.333333333333336

Process finished with exit code 0
```

Test case 2

Processes with these burst_times and priorities are arrived to the queues at the current_time=0

Process ID	Burst Time	Priority
1	25	0
2	7	0
3	3	0
4	2	1
5	7	1
6	10	1
7	4	2
8	6	2
9	11	2
10	11	3
11	7	3
12	5	3

All these processes are inserted to the queues at the same arrival_time of 0 sec. Then the queues will be like this.

Front **Rear**

25->20->15 p1	7->2->0 p2	3->0 p3	Q0
2->0 p6	7->0 p5	10->0 p4	Q1
4->0 p9	6->0 p8	11->1 p7	Q2

11->0 p10	7->0 p11	5->3 p12	Q3

P3, p2, p4, p5, p6, p7, p8, p10, p11, p1, p9, p12

```
Process id : 1 takes CPU to execute now at 0
Process id : 2 takes CPU to execute now at 5
Process id : 3 takes CPU to execute now at 10
The process with process id 3 completed successfully at 13
turndaround time for 3 is : 13
waiting time for 3 is : 10

Process id : 1 takes CPU to execute now at 13
Process id : 2 takes CPU to execute now at 18
The process with process id 2 completed successfully at 20
turndaround time for 2 is : 20
waiting time for 2 is : 13

Process id : 1 takes CPU to execute now at 20
Process id : 1 takes CPU to execute now at 25
Process id : 1 takes CPU to execute now at 30
The process with process id 1 completed successfully at 35
turndaround time for 1 is : 35
waiting time for 1 is : 10

Process id : 4 takes CPU to execute now at 35
The process with process id 4 completed successfully at 37
turndaround time for 4 is : 37
waiting time for 4 is : 35

Process id : 5 takes CPU to execute now at 37
Process id : 5 takes CPU to execute now at 40
The process with process id 5 completed successfully at 44
turndaround time for 5 is : 44
waiting time for 5 is : 37
```

```
Process id : 6 takes CPU to execute now at 44
The process with process id 6 completed successfully at 54
turndaround time for 6 is : 54
waiting time for 6 is : 44

Process id : 7 takes CPU to execute now at 54
The process with process id 7 completed successfully at 58
turndaround time for 7 is : 58
waiting time for 7 is : 54

Process id : 8 takes CPU to execute now at 58
Process id : 8 takes CPU to execute now at 60
The process with process id 8 completed successfully at 64
turndaround time for 8 is : 64
waiting time for 8 is : 58

Process id : 9 takes CPU to execute now at 64
The process with process id 9 completed successfully at 75
turndaround time for 9 is : 75
waiting time for 9 is : 64

Process id : 10 takes CPU to execute now at 75
Process id : 10 takes CPU to execute now at 80
The process with process id 10 completed successfully at 86
turndaround time for 10 is : 86
waiting time for 10 is : 75

Process id : 11 takes CPU to execute now at 86
The process with process id 11 completed successfully at 93
```

```
Process id : 12 takes CPU to execute now at 93
The process with process id 12 completed successfully at 98
turndaround time for 12 is : 98
waiting time for 12 is : 93

Processes successfully executed.............................

Total waiting time and turn around time calculated

Total waiting time for roundrobin : 33
Total TurnAround time for roundrobin : 68
Total waiting time for SJF1 : 180
Total TurnArond time for SJF1 : 210
Total waiting time for SJF2 : 112
Total TurnArond time for SJF2 : 122
Total waiting time for FCFS : 254
Total TurnArond time for FCFS : 277

Average waiting time and turn around time calculated

Average waiting time for roundrobin : 11.0
Average TurnAround time for roundrobin : 22.666666666666668
Average waiting time for SJF1 : 60.0
Average TurnArond time for SJF1 : 70.0
Average waiting time for SJF2 : 37.333333333333336
Average TurnArond time for SJF2 : 40.666666666666664
Average waiting time for FCFS : 84.66666666666667
Average TurnArond time for FCFS : 92.33333333333333
```

To compare waiting time and turnaround time in 4 queues I entered same clones of processes set to each queues.

Processes with these burst_times and priorities are arrived to the queues at the current_time=0

Process ID	Burst Time	Priority
1	10	0
2	5	0
3	2	0
4	10	1
5	5	1
6	2	1
7	10	2
8	5	2
9	2	2
10	10	3
11	5	3
12	2	3

All theses processes are inserted to the queues at the same arrival_time of 0 sec. Then the queues will be like this.

Front **Rear**

10->5->0 p1	5->0 p2	2->0 p3	Q0

2->0 p6	5->0 p5	10->0 p4	Q1

2->0 p9	5->0 p8	10>0 p7	Q2

10->0 p10	5->0 p10	2->0 p10	Q3

Process should be completed like this.

P2, p3, p1, p6, p5, p4, p9, p8, p7, p10, p11, p12

```
Process id : 1 takes CPU to execute now at 0
Process id : 2 takes CPU to execute now at 5
The process with process id 2 completed successfully at 10
turndaround time for 2 is : 10
waiting time for 2 is : 5

Process id : 3 takes CPU to execute now at 10
The process with process id 3 completed successfully at 12
turndaround time for 3 is : 12
waiting time for 3 is : 10

Process id : 1 takes CPU to execute now at 12
The process with process id 1 completed successfully at 17
turndaround time for 1 is : 17
waiting time for 1 is : 7

Process id : 6 takes CPU to execute now at 17
The process with process id 6 completed successfully at 19
turndaround time for 6 is : 19
waiting time for 6 is : 17

Process id : 5 takes CPU to execute now at 19
The process with process id 5 completed successfully at 24
turndaround time for 5 is : 24
waiting time for 5 is : 19

Process id : 4 takes CPU to execute now at 24
The process with process id 4 completed successfully at 34
turndaround time for 4 is : 34
waiting time for 4 is : 24
```

```
Process id : 9 takes CPU to execute now at 34
The process with process id 9 completed successfully at 36
turndaround time for 9 is : 36
waiting time for 9 is : 34

Process id : 8 takes CPU to execute now at 36
The process with process id 8 completed successfully at 41
turndaround time for 8 is : 41
waiting time for 8 is : 36

Process id : 7 takes CPU to execute now at 41
The process with process id 7 completed successfully at 51
turndaround time for 7 is : 51
waiting time for 7 is : 41

Process id : 10 takes CPU to execute now at 51
The process with process id 10 completed successfully at 61
turndaround time for 10 is : 61
waiting time for 10 is : 51

Process id : 11 takes CPU to execute now at 61
The process with process id 11 completed successfully at 66
turndaround time for 11 is : 66
waiting time for 11 is : 61

Process id : 12 takes CPU to execute now at 66
The process with process id 12 completed successfully at 68
turndaround time for 12 is : 68
waiting time for 12 is : 66
```

```
Processes successfully executed.........................

Total waiting time and turn around time calculated

Total waiting time for roundrobin : 22
Total TurnAround time for roundrobin : 39
Total waiting time for SJF1 : 101
Total TurnArond time for SJF1 : 128
Total waiting time for SJF2 : 70
Total TurnArond time for SJF2 : 77
Total waiting time for FCFS : 178
Total TurnArond time for FCFS : 195

Average waiting time and turn around time calculated

Average waiting time for roundrobin : 7.333333333333333
Average TurnAround time for roundrobin : 13.0
Average waiting time for SJF1 : 33.666666666666664
Average TurnArond time for SJF1 : 42.666666666666664
Average waiting time for SJF2 : 23.333333333333332
Average TurnArond time for SJF2 : 25.666666666666668
Average waiting time for FCFS : 59.333333333333336
Average TurnArond time for FCFS : 65.0

Process finished with exit code 0
```

First Come First Serve

Pros-

- Easy to emplement
- Time complexity is lower
- Simple
- User friendly

Cons-

- Long Waiting Time for the small processes if it arrives at last
- Lower Device Utilization
- No ideal time sharing system

Shortest Job First

Pros-

- Easy to emplement than round robin
- Short Waiting Time
- Simple than round robin
- User friendly

Cons-

- No ideal time sharing system
- Turn Around time for the big processes are longer as it waits until all the short processes to execute even it arrives at the first.

Round Robin

Pros-

- Waiting time is likely equal for all processes
- Performs well if the burst times are low (<time quantum)
- Reduces starvation because it shares times with the processes of queue

Cons-

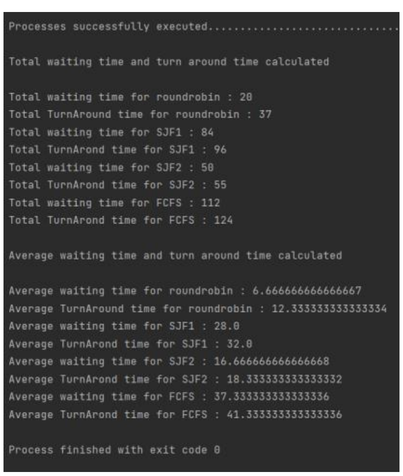
- Hard to implement
- Queue take a lot time if burst times are higher.
- Waiting time can be longer than shortest job first

Because all the processes arrives at current_time 0 , Turn around time is the current_time when a process completes its execution. Waiting time is turn around time- burst time. The time a process waits before exexcution.

Total waiting time is total of the waiting times in processes at the queue.

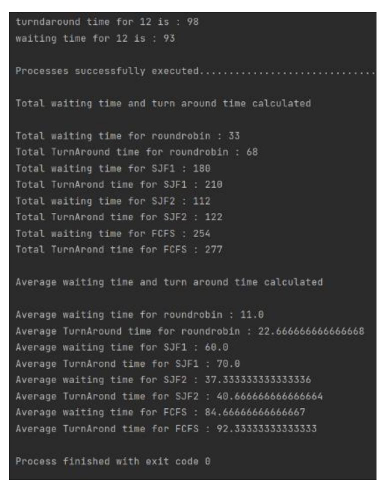
Total turn around time is total of the turn around times in processes at the queue.

In first test case



```
Processes successfully executed...............................

Total waiting time and turn around time calculated

Total waiting time for roundrobin : 20
Total TurnAround time for roundrobin : 37
Total waiting time for SJF1 : 84
Total TurnArond time for SJF1 : 96
Total waiting time for SJF2 : 50
Total TurnArond time for SJF2 : 55
Total waiting time for FCFS : 112
Total TurnArond time for FCFS : 124

Average waiting time and turn around time calculated

Average waiting time for roundrobin : 6.666666666666667
Average TurnAround time for roundrobin : 12.333333333333334
Average waiting time for SJF1 : 28.0
Average TurnArond time for SJF1 : 32.0
Average waiting time for SJF2 : 16.666666666666668
Average TurnArond time for SJF2 : 18.333333333333332
Average waiting time for FCFS : 37.333333333333336
Average TurnArond time for FCFS : 41.333333333333336

Process finished with exit code 0
```

in the second test case



```
turndaround time for 12 is : 98
waiting time for 12 is : 93

Processes successfully executed...............................

Total waiting time and turn around time calculated

Total waiting time for roundrobin : 33
Total TurnAround time for roundrobin : 68
Total waiting time for SJF1 : 180
Total TurnArond time for SJF1 : 210
Total waiting time for SJF2 : 112
Total TurnArond time for SJF2 : 122
Total waiting time for FCFS : 254
Total TurnArond time for FCFS : 277

Average waiting time and turn around time calculated

Average waiting time for roundrobin : 11.0
Average TurnAround time for roundrobin : 22.666666666666668
Average waiting time for SJF1 : 60.0
Average TurnArond time for SJF1 : 70.0
Average waiting time for SJF2 : 37.333333333333336
Average TurnArond time for SJF2 : 40.666666666666664
Average waiting time for FCFS : 84.66666666666667
Average TurnArond time for FCFS : 92.33333333333333

Process finished with exit code 0
```

conclusion, summarizing the findings of the analysis and discussing the limitations of the program

conclusion and summarizing the findings of the analysis

FCFS scheduling has a longer waiting time than others. But the implementation is simpler. Shortest Job first scheduling has shorter waiting time than first come first serve. But those two scheduling algorithms are not suitable for time sharing systems like foreground processes. These are suitable for background processes. Processes at the front take lower waiting time and processes at back have longer waiting time. Round Robin scheduling algorithms is much suitable for time sharing systems. It can share cpu with all the processes in roundrobin queue. Cpu switches between processes by time quantum of time.

This multilevel queue enables cpu to switch between all the queues by 20 seconds of timer. It will enable time sharing between queues. If not time share , the processes at the FCFS will be starved until the processes at above queues are completed successfully.

When same set of processes were given to the each queues roundrobin algorithm executed poorly compared with other queues. While the SJF queues were fairly average and the FCFS algorithm performed a bit better than the RoundRobin.

limitations

I have created this program for processes which have same arrival time of 0. So preemption does not happens. If these processes have different arrival times we should consider the time processes arrives and taking cpu by higher queues always. Implementing of such a program is not easy. It will be harder to implement and thinking the algorithem.

If user inputs invalid user inputs as error handling has not been done there may be a chance of getting errors. So user should input valid inputs everytime.

Process id s are generated automatically in ascending order.

Implementing this by C or C++ will also be tougher than this. We have to creates queues. Here we can use lists in python. And sorting like thinks are easier here. Large number of codes are taken for the printing stuffs.

2021/CS/198

21001987

W.H.K.Udara