round robin algorithm

|  |
| --- |
|  |
|  |
| **‏** |

Teacher: Mohammed Al-Musa'idin

Student: Hassan Al-Omari

University number: 320220603072

Introduction

*Implementation details: Enter the process names, enter the quantum number, and enter the remaining burst time like this: avg(remBt, q, wt, process).*

*The programming language is Python.*

*I did not use any tools.*

1. Effect of Time Quantum size on performance: If the quantum is too small: context switch rates increase, if the quantum is too large: behavior approaches FCFS algorithm.
2. Priority Ignorance.
3. Waiting time may be higher on average for some loads.

Disadvantages (RR)

1. Ease of understanding and application.
2. Reduce average waiting time.
3. Starvation-Free.
4. Good Responsiveness.
5. Flexible and easy to adjust.

(RR) Features

1. Fairness: Ensuring that each process gets an equal share of processing time in each cycle.
2. Response Time: Less than others.
3. Starvation Prevention Reduce.
4. Ease of implementation.
5. High Interactivity.

(RR) Properties

*The basic idea of ​​the Round Robin algorithm (abbreviated as RR) is to allocate short time periods (Time Quantum) to each process in a circular order (circular in the form of a list). When the time allocated to a process ends without it being completed, the process is returned to the end of the queue (Ready Queue) and the processing unit moves on to executing the next process in the cycle.*

Function waiting\_time(remBt, q, waitTime, completeTime, p, g):

# remBt: List of remaining burst times for each process

# q: Quota time

# waitTime: List to be populated with waiting times for each process

# completeTime: List to be populated with completion times for each process

# p: List of process IDs in the same order as remBt

# g: Initially, an empty list, to which we will add the process ID for each time cycle to build the Gantt chart

1. Create a copy of the remaining burst times (rBt) for later calculations:

rBt ← Copy the contents of remBt

2. Set the variable time = 0 # to represent the current total time since the start of the schedule

3. Repeat until all processes have finished:

repeat:

final ← true # Flag indicating that all processes have finished (will be reflected (Later, if there is work remaining)

# Review each process i

For each process i from 0 to (number of processes − 1):

If (remBt[i] > 0): # If work i has non-zero execution time remaining, it has not finished yet

final ← false # At least one process has not finished yet

If (remBt[i] > q): # If work has more time remaining than the quota q:

time ← time + q # Execute the work in q time units

remBt[i] ← remBt[i] − q # Decrease the remaining time for the work

Add p[i] to the list g # Record the running process in the Gantt chart

Else: # If work has time remaining ≤ q, it will finish in this iteration

time ← time + remBt[i] # Execute the work for the remainder of its duration

completeTime[i] ← time # Completion time for work i

waitTime[i] ← time − rBt[i] # Wait time = (Completion time − Original execution time)

remBt[i] ← 0 # Remaining execution time becomes zero

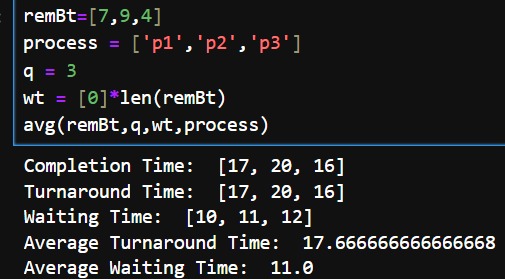
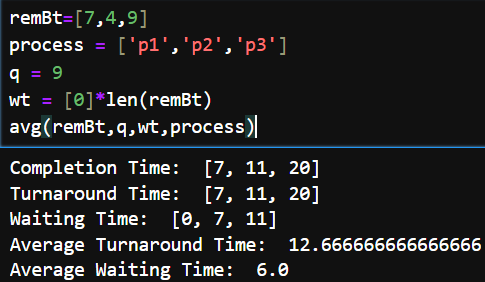
Add p[i] to list g # Record the running process in the Gantt chart

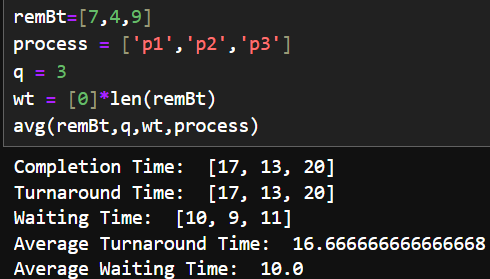
if (final == true): # If no process is found, remBt[i] > 0

Exit the loop (all processes have finished)

4. The function terminates, and waitTime , completeTime , and g are filled in.

[Cite your source here.]

A computer screen shot of a program

AI-generated content may be incorrect.

Case 2

Case 4

Here, we notice that in cases 1 through 3, we only change the order of the remaining burst time list, which changes the average waiting time and average turnaround time. We also notice that the values ​​decrease as we sort the remaining burst time list, placing the smallest values ​​first, and increase as we sort the list, placing the largest values ​​first. This is because small values ​​complete their work quickly, while large values ​​take time to complete their work.

We also notice that when we change q to a larger value, the algorithm works more like FCFS.

Case 4

Case 3

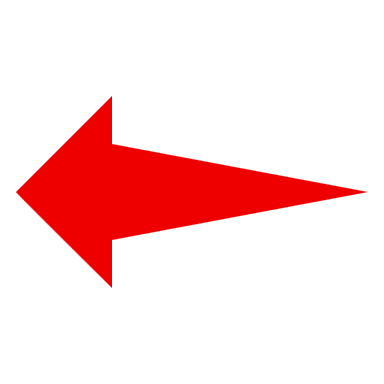
Case 2

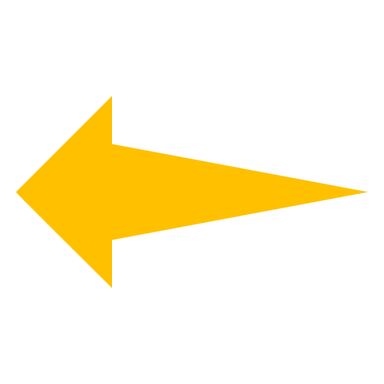
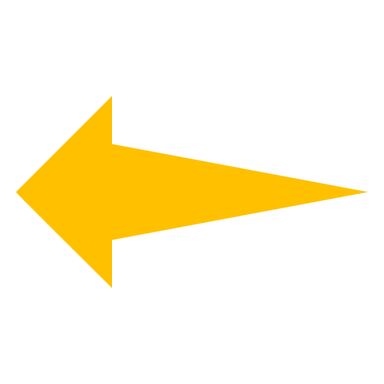
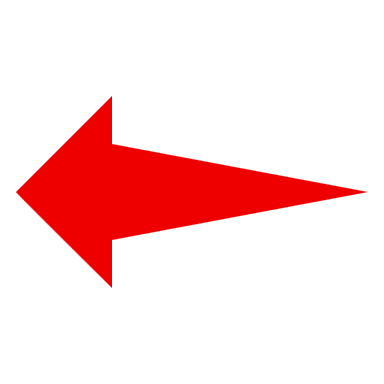
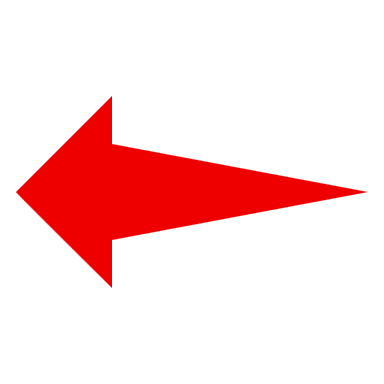
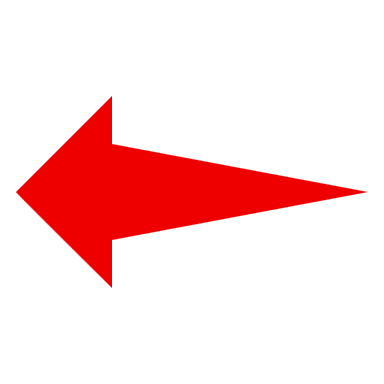
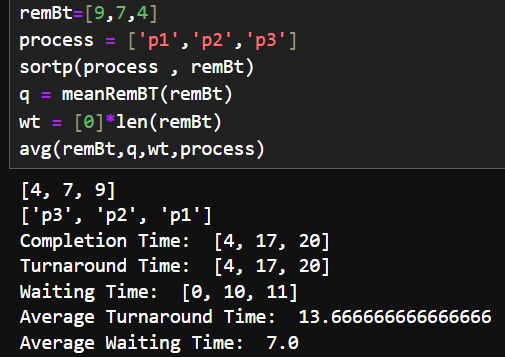
Case 1

Case 3

Case 1

Test results

A computer screen shot of a program

AI-generated content may be incorrect.

After

Before

Before

After

As we've seen, the order of operations clearly affects the results.

I made a simple improvement to the algorithm by ordering the operations before entering them into the algorithm.

As we've observed that the value of q affects the results, I filled it with the average of the remaining burst time.

The algorithm now performs better than before, significantly reducing context switching.

Conclusion and Improve