Round Robin CPU Scheduling Project Report

CSCI330-CS-2023SP-S

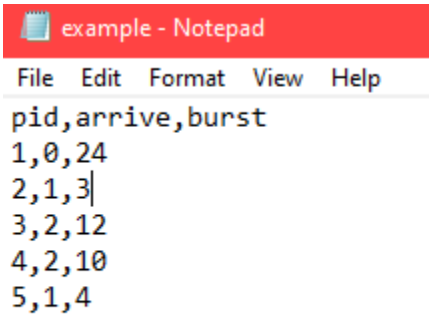
By: Daniel Yurskiy

Whenever a process needs to be run, it needs power from the CPU in order to carry out the tasks it is designed to carry out. However, when there are many processes wanting to run, we cannot run those processes all at the same time. Not only that, but if we are running a process that is a heavy load to the cpu, there can be cases where some processes will be infinitely stuck waiting for their time to run when they only need to run for a very small amount of time . Therefore, we need to create an algorithm that correctly picks which processes to run and which processes to give the correct amount of power so that we can max out the cpu utilization and minimize the amount of waiting time for other processes. These algorithms are called CPU scheduling algorithms and consist of algorithms like “First-Come-First-Serve” and “Priority Scheduling” those of which are algorithms that run the first process entered into a ready queue and run processes based on their priority given by the creator of the processes respectively. However in this project, we are going to be focusing on the “Round-Robin” scheduling algorithm. This algorithm consists of you giving the algorithm a specific amount of time you would like, aka a time quantum, where once a process reaches the ready queue, it will run that process at the desired time quantum you gave it. Each process will have its own arrival time, and its own burst time, arrival time being when the process should arrive at the ready queue and begin executing and burst time being how long a process has left until it finishes. When it comes to running a process from ready queue I will use the FIFO scheduling algorithm which runs the first process inserted into the queue.

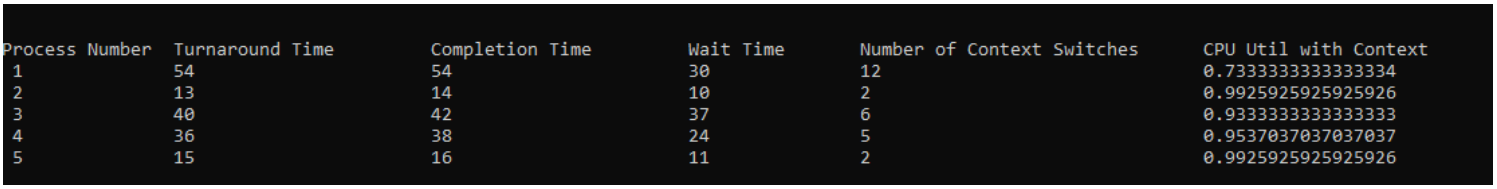
In my program written to simulate the round robin process you are able to input the name of a file that contains all of the information regarding an individual process and your desired time quantum. With this info I store all of the corresponding information into a variable created from my Processes class called “our_prosseses”. My Processes class stores all the information about a

process that I would need like the amount of context switches that a certain process does or a certain process's completion time. I then have a dispatcher class that stores all the information needed for analyzing the queues needed in round robin like the ready queue and the waiting queue. After moving all of the desired information into its respective arraylists using my Process class, I then got to creating a method called initalizeReadyqueue that fills the ready queue with processes sorted by their arrival time. After that was created, I got to developing the round robin algorithm with a method that is aptly named RoundRobin. Before getting into the calculations of each process, I not only needed to initialize the ready queue using my initalizeReadyqueue function, but I needed to make sure that the arrival time of the program matched the first arrival time of the process in the ready queue, after that then I got to making the logic of running the process. I created this logic by comparing the burst time of the first process currently in the ready queue with the time quantum and checking if it is less than the time quantum, greater than the time quantum or the same as the time quantum. No matter what the result is of that logic, I increment the current time value I have with the time quantum, as regardless of the result, you are still running a process for that time quantum. If the burst time of a process is less than the time quantum or the same as the time quantum, then that means after the process runs for that time quantum, then it is finished executing, therefore I send the process into a terminate queue that removes the process from its respective arraylists. Then I do specific calculations to store the corresponding waiting time, turn around time and completion time for each process based on if the process is less than, greater than or equal to the time quantum. Then my round robin function finishes after the processes in the ready queue have iterated once, after they have iterated I have a while loop that checks if there are any processes left and if they are then I call my round robin function. In my main function, I start the round robin process and print all the useful information.

When it comes to printing useful information, my program prints what is happening at what times during the round robin function and at the end of round robin, it prints out a table of all of the processes with their respective completion time, average wait time, number of context switches and their cpu utilization.

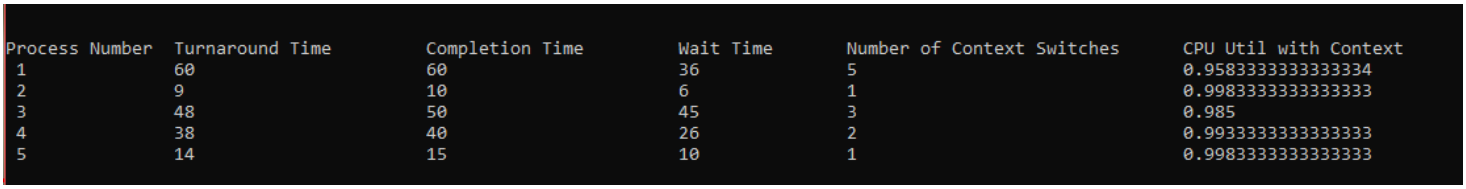


- These are the process values I will be using for my testing, and my file name will be “example” however your file name does not have to be “example”.



Process Number	Turnaround Time	Completion Time	Wait Time	Number of Context Switches	CPU Util with Context
1	54	54	30	12	0.7333333333333334
2	13	14	10	2	0.9925925925925926
3	40	42	37	6	0.9333333333333333
4	36	38	24	5	0.9537037037037037
5	15	16	11	2	0.9925925925925926

- With a time quantum of 2, this is what my program outputted.



Process Number	Turnaround Time	Completion Time	Wait Time	Number of Context Switches	CPU Util with Context
1	60	60	36	5	0.9583333333333334
2	9	10	6	1	0.9983333333333333
3	48	50	45	3	0.985
4	38	40	26	2	0.9933333333333333
5	14	15	10	1	0.9983333333333333

-With a time quantum of 5, this is what my program outputted.

Process Number	Turnaround Time	Completion Time	Wait Time	Number of Context Switches	CPU Util with Context
1	80	80	56	3	0.98875
2	19	20	16	1	0.99875
3	68	70	65	2	0.995
4	38	40	28	1	0.99875
5	29	30	25	1	0.99875

-

- With a time quantum of 10, this is what my program outputted, notice how the cpu util not only stays above 90% but as the number of context switches decreases significantly, then the cpu util also reaches closer and closer to 100%.

Process Number	Turnaround Time	Completion Time	Wait Time	Number of Context Switches	CPU Util with Context
1	53	53	29	24	-0.0867924528301891
2	11	12	8	3	0.9830188679245283
3	39	41	36	12	0.7283018867924529
4	35	37	23	10	0.8113207547169812
5	16	17	6	4	0.969811320754717

-With a time quantum of 1, this is what my program outputted, obviously it is impossible that a cpu can be utilized to a negative percentage, but we can assume that it is still a very low amount. Compared to the time quantum being 10, you can notice that the cpu utilization has significantly decreased as has the amount of context switches.

Process Number	Turnaround Time	Completion Time	Wait Time	Number of Context Switches	CPU Util with Context
1	24	24	0	1	0.9991666666666666
2	47	48	44	1	0.9991666666666666
3	118	120	106	1	0.9991666666666666
4	94	96	84	1	0.9991666666666666
5	71	72	67	1	0.9991666666666666

- With a time quantum of 24, this is what my program outputted, even though the cpu utilization is at 99% this time quantum is worse than other time quantums because of the

wait time and the time it took to complete each process being way higher than it should be.

This is an example of what my program outputs to the console when using the time quantum of 24

```
C:\Users\DanDaMan\Desktop\src>java target example.txt 24
Process 1 added to ready queue
Process 2 added to ready queue
Process 5 added to ready queue
Process 4 added to ready queue
Process 3 added to ready queue
Processes Left = [1, 2, 3, 4, 5]
Ready Que = [1, 2, 5, 4, 3]
Terminating process 1 at time 24
Terminating process 2 at time 48
Terminating process 5 at time 72
Terminating process 4 at time 96
Terminating process 3 at time 120
proccesses = [] processes arrive time = [] processes burst time = []
Total Context switches 5
5 total processes removed
```

And to show that it doesn't only output once here is what my program outputs when using the time quantum of 5.

```
C:\Users\DanDaMan\Desktop\src>java target example.txt 5
Process 1 added to ready queue
Process 2 added to ready queue
Process 5 added to ready queue
Process 4 added to ready queue
Process 3 added to ready queue
Processes Left = [1, 2, 3, 4, 5]
Ready Que = [1, 2, 5, 4, 3]
Terminating process 2 at time 10
Terminating process 5 at time 15
proccesses = [1, 3, 4] processes arrive time = [0, 2, 2] processes burst time = [19, 7, 5]
Total Context switches 5
2 total processes removed

Process 1 added to ready queue
Process 3 added to ready queue
Process 4 added to ready queue
Processes Left = [1, 3, 4]
Ready Que = [1, 3, 4]
Terminating process 4 at time 40
proccesses = [1, 3] processes arrive time = [0, 2] processes burst time = [14, 2]
Total Context switches 8
3 total processes removed
```