EE5903 Real Time System CA1

(Group 20)

Variants of Round Robin CPU Scheduling Algorithm Based on Dynamic Quantum Time

Done By:

A0169907M Emil Yong Kai Wen

A0232587X Kallur Nitin Gururaj

Abstract

In scheduling of CPU tasks, there are various algorithms which try to solve the problem of optimal scheduling. One of the most popular CPU scheduling algorithms is the Round Robin (RR) algorithm. However, RR is significantly dependent on the choice of Quantum Time (QT) used. In classical RR, it uses fixed QT. For a small quantum, it is time consuming, offers low throughput and larger waiting and turnaround time. In this paper, various proposed algorithms are presented with Dynamic Quantum Time for optimal Average Waiting Time (AWT), Average Turnaround Time (ATT) and starvation based on different task arrival times.

Introduction

Operating system interacts between the user and the computer hardware. Its sole purpose is to provide a platform in which a user can execute programs in a well-located and efficient manner. Modern operating system and time-sharing systems are becoming more advance and complex. They have evolved from a single task to a multitasking environment in which processes run in a synchronized manner. Therefore, CPU scheduling is one of the basic features of real time system and is necessary for any operating system task [1]. With the current COVID-19 pandemic on going, allocation of vaccines poses a huge challenge to populated countries such as India.[5] The current ranking system is based on the concept that health care person should be given the priority followed by the frontliner, seriously ill patient, elderly and then the public. Hence, first come first serve basis of scheduling is not practical. Getting an optimized CPU scheduling algorithm for Vaccine scheduling is vitally important because in India, the current algorithm used was not efficient enough when the second wave of COVID-19 hit. Many lives were lost even though vaccines were available. However, because of the poor scheduler used, no one was taking them and even if people started to register, only a handful of the citizen are given the chance to register in a small window time based on the First Come First Serve approach. [5].

Therefore, in order to optimize the number of people to receive the vaccine, it is essential to have an optimized CPU scheduling algorithms where different CPU scheduling methods are analyzed based on Arrival Time, Turn Around Time, waiting and response time for Vaccine scheduling.

Currently, there are many different CPU scheduling algorithms by which a resource is allocated to a process or task and executes in different ways. For scheduling, there are FCFS, SJF, RR etc. These processes are scheduled according to the given burst time and arrival time. However, in this paper, the primary CPU scheduling technique discussed, is the variants of Round Robin; *EDRR*, *IDRR*, *MDRR* and *ESRR*. This paper will find the best Round Robin variant that optimize average waiting time and average turnaround time.

Context Switching time is a desirable feature of time-sharing system. However, in CPU scheduling it is undesirable. The proposed algorithm needs to avoid unnecessary context switching as much as possible to improve its efficiency. It can be achieved by choosing a suitable time quantum which lowers the number of processes and thereby reducing the context switching [3]

The context switching time can be calculated as follows:[3]

$$Context\ Switching\ time\ = \{\sum P_i\} - \ 1 \quad (1)$$

Average Waiting Time (AWT) of a scheduling algorithm is an important metric, as it signifies the average amount of time a process has to wait to execute. An optimal scheduling algorithm would have low average waiting time, as this would mean a process would have to wait less time to execute. Average waiting time can be calculated as follows:

$$Avg\ Waiting\ Time\ (AWT) = \{\sum W_{T_i}\}/n \quad (2)$$

Average Turnaround Time (ATT) of a scheduling algorithm is another metric which is of prime importance, as it denotes the response time for a process to be completed once it arrives for execution. Having a lower ATT also indicate a higher algorithm efficiency. The ATT can be calculated as follows:

$$Avg\ \ Turn\ around\ \ Time\ (ATT) = \{\sum T_{T_i}\}/n \quad (3)$$

Discussion of algorithms

In this section, we study four algorithms from four papers: *EDRR*, *IDRR*, *MDRR* and *ESRR*. All four papers try to propose methods to solve the problem of optimal scheduling for CPU scheduling of processes. The common strategy used in all these four papers is Round Robin (RR) scheduling algorithm. But they apply different approaches of Round Robin to optimize the scheduling problem. We compare the four algorithms based on the following aspects:

Quantum Time (QT) Used: As it is well known that the RR algorithm's performance depends heavily on the chosen quantum time. If the quantum time chosen is very small, it will lead to unnecessary context switching, and if it very large, the algorithm degenerates to FCFS algorithm. In IDRR, the approach to selecting the time quantum is based on taking the average of highest burst times and subtracting the average of the two lowest arrival time. In EDRR however, the selected time quantum is a factor multiplied with the max burst time. Whereas, MDRR proposes to use Variable Time Quanta (VTQ) for different processes. Lastly, ESRR does not provide a criterion for choosing the quantum time, and it can be chosen randomly which is in stark contrast with rest of the algorithms.

Arrival Time Consideration: Arrival Times of processes play an important role in the performance of the algorithm. Some algorithms choose to tackle the problem of different arrival times, but some choose to ignore it and assume same arrival times for all the processes. EDRR, IDRR and ESRR address the problem of different and same arrival times whereas MDRR addresses only the same arrival time problem. MDRR assumes that all the processes are available in the ready queue at the beginning of execution.

Starvation Problem: Starvation can be defined as a process being denied access to the CPU for execution due to poor schedule generated by a scheduling algorithm. It is essential for scheduling algorithms to allocate CPU to every process in the ready queue to avoid starvation and hence prevent longer AWT and ATT. MDRR, ESRR and IDRR address this problem by assigning the CPU to longer processes whereas EDRR does not consider this issue.

Round Robin (RR)

Round Robin scheduling is a preemptive algorithm which is effective in time-sharing environments. RR assigns a constant quantum time for each process in the ready queue. When the QT terminates, the current process is added to the end of the ready queue if it does not finish execution.

Algorithm includes the following steps [3]: **Step1**: Select time quantum and assign to all process. **Step2**: Assign the CPU to processes in the ready queue in a FCFS manner. **Step3**: If burst time is longer than time quantum, go to step 4. Else, go to step 5. **Step4**: Process is executed till the time quantum and its remaining burst time is inserted at the end of ready queue for next execution. **Step5**: Process is allocated to CPU until it finishes. **Step6**: Go to step 2 until ready queue is empty. RR is said to be better than other scheduling algorithms because it reduces the turnaround time and waiting time.

Efficient Dynamic Round Robin (EDRR)

As aforementioned, the efficiency of Round Robin is highly dependent on the time quantum selected. With the advancement of technology, many different techniques can be implemented to dynamically control the time quantum. Therefore, in many Dynamic Round Robin (DRR) algorithms, the time quantum is dynamically chosen instead of using a constant value. However, one main problem in most DRR algorithm is the cleverness of choosing this time quantum.

Different applications have different set of tasks' burst time and arrival time. Hence if an unwise choice of time quantum is selected, there is a possibility that the specific task has to wait for all other processes to complete even if the remaining execution time is 1- or 2-unit time. This phenomenon can snowball and affect the average waiting time.

EDRR [1] is proposed to tackle this issue where it selects an elastic dynamic time quantum that will allow a process to execute completely if its remaining execution time is lesser than 0.2^{th} of the initial time. This is said to ensures higher efficiency

and lesser context switches, average waiting and turn around time with high through-put. In EDRR, the time quantum is chosen as 0.8^{th} times the maximum burst time of a process in the ready queue. After which, for all the processes in the ready queue that has the burst time lesser than the selected time quantum, it will be send to the CPU for execution. During which, the task which has larger time quantum than 0.8^{th} of the maximum time quantum will wait in the ready queue until all the tasks with smaller burst size has been sent to the CPU for execution.

When all the smaller tasks have completed, the time quantum is updated back to the maximum burst time. This will allow sufficient time for all the processes to complete without having to wait for the remaining process to complete when it has small unit of execution time remaining. The flow chart is shown in Figure 1.

Where:
I = iteration number
Max_b_time = maximum burst time
Task_i_b_time = task burst time
NT/N = number of process/tasks
QT = Quantum time

Figure 1 EDRR Flowchart

In addition, EDRR is also capable to handle processes arriving in the ready queue at different timing, the algorithm will re-calculate the time quantum at the end of each quantum. This is also shown on the flow chart above.

Next, below are two examples to help us understand the proposed algorithm. In each example, a Gantt chart is drawn to illustrate the time quantum and waiting time for each process. The two examples shown are with different and same arrival time.

Same Arrival Time (EDRR)

ID	Arrival Time (BT)	Burst time (BT)	Turnaround time	Waiting Time (WT)
P1	0	11	27-0 = 27	16
P2	0	10	10-0 = 10	0
P3	0	6	16-0=16	10
P4	0	13	40-0 = 40	27

Table 1: Same arrival time (EDRR)

Based on the table above, the maximum burst time is at process ID 4 which has a burst time of 13. Hence using the EDRR algorithm, the first quantum time equals to 0.8 * 13 = 10.4. This means that out of the above 4 processes, only P2 and P3 will be sent to CPU. Meanwhile, P1 and P4 will be placed at the end of the ready queue because their burst time is larger than quantum time 10.4. Once P2 and P3 have completed its execution, the quantum time will be set to Max_b_time which is 13. P1 and P4 can now be assigned to the CPU.

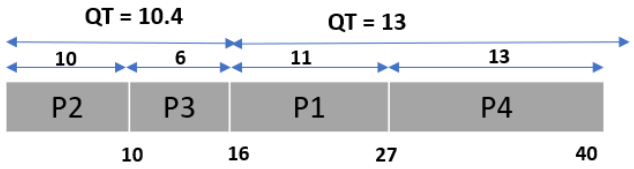


Figure 2: Same arrival time (EDRR) Gantt chart

$$Average\ waiting\ time\ (AWT) = \frac{53}{4} = 13.25$$

$$Average\ Turnaround\ time\ (ATT) = \frac{93}{4} = 23.25$$

Different Arrival Time (EDRR)

ID	Arrival Time (BT)	Burst time (BT)	Turnaround time	Waiting Time (WT)
P1	0	35	35-0 = 35	0

P2	6	70	193-6 = 187	117
P3	9	50	85-9=76	26
P4	15	38	123-15 = 108	70

Table 2: Different arrival time (EDRR)

The above table shows processes arriving at different arrival time. According to the algorithm, EDRR looks only at the processes in the ready queue. At time, t=0, only P1 is in the ready queue. Therefore, the Max_b_time = 35 and time quantum, QT = 0.8 * 35 = 28. P1 will have a remainder of 7-unit execution time. However, while P1 is executing in the 35-unit time span, all the other processes will arrive in the ready queue. Now, the Max_b_time is = 70, time quantum = 70 * 0.8 = 56. Based on the EDRR, since P2 > 56, it will be placed at the end of the ready queue. P1 will continue to execute as it is lower than 56. Afterwards, P3, P4 and P2 are assigned CPU. The Gannt chart is shown below.

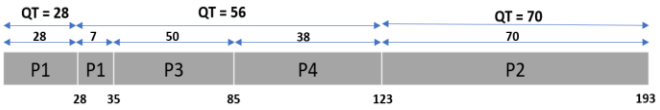


Figure 3: Different arrival time (EDRR) Gantt chart

$$Average\ waiting\ time\ (AWT) = \frac{213}{4} = 53.25$$

$$Average\ Turnaround\ time\ (ATT) = \frac{406}{4} = 101.5$$

An Improved Dynamic Round Robin (IDRR)

In this second algorithm, the improved dynamic round robin (IDRR) controls and computes the time quantum for Round Robin in another way. Setting too large quantum time is said to increase the average waiting time and turn-around times. In many real time systems with aperiodic tasks, it is required to be processed as fast as possible, which in return, means fast response time. IDRR is an improved version of the above-said round robin method [2], to improve the delay by around 40-55%. In addition, with IDRR, it is also said that the number of overheads occurring from context switching can be minimized.

Since Round Robin scheduling time quantum plays a significant role to improve the performance, IDRR makes sure that the chosen quantum time is as large as possible. Therefore, the time quantum for same arrival time task is chosen by taking the average of the two largest burst time. After which, for the remaining task, the time quantum is updated to the average of largest remaining task execution time with the old or previous quantum time used.

Whereas, for different arrival time tasks, IDRR will immediately assign the first ready task in the ready queue to the CPU immediately. The first-time quantum will be based on the first ready task's burst time. During the execution of the first task, several other tasks may arrive in the ready queue.

The quantum time will be updated in the next round, once the first task has completed. It subtracts the average of the highest remaining burst time plus the old/previous quantum time with the average of the two lowest remaining arriving time for once.

With the updated quantum time, it will process all the task to the CPU. For the subsequent rounds, the quantum time is updated by subtracting the average of the highest remaining burst time with the old/previous quantum time and the lowest remaining arrival time /2. The flow chart of the above mentioned is shown in Figure 4.

Same Arrival Time (IDRR)

ID	Arrival Time (AT)	Burst Time (BT)	Turnaround time	Waiting Time (WT)
P3	0	6	6-0 = 6	0
P2	0	10	16-0 = 16	6
P1	0	11	27-0=27	16
P4	0	13	40-0 = 40	27

Table 3: Same arrival time (IDRR)

First and foremost, all the processes are arranged in the ascending order. For the first iteration, the time quantum is calculated as (11+13) / 2 = 12. Therefore, Process 1, 2 and 3 with the burst time of 11, 10 and 6 respectively will be executed fully. However, for P4, there will be a remaining of 1 unit execution time after the first time quantum. In the next round, the time quantum is update to (1+ 12) /2 = 6.5. This updated time quantum will be assigned for P4 which has a remaining on 1 unit execution time.

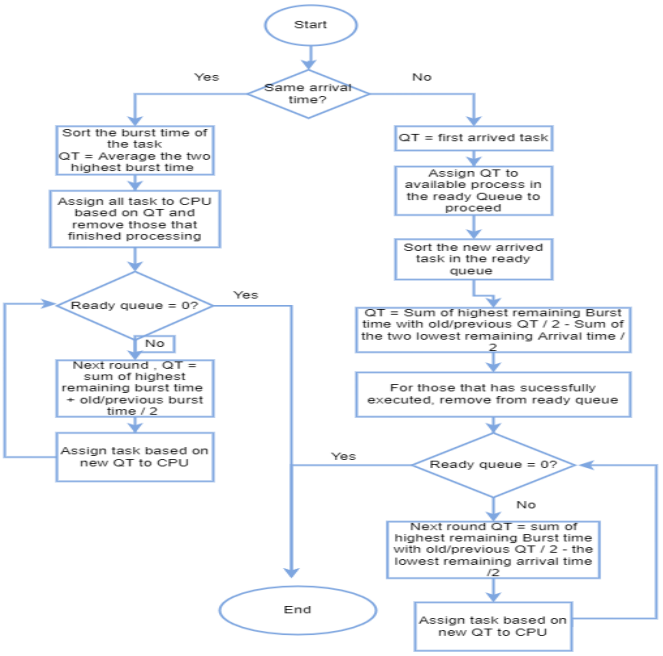


Figure 4: 2 IDRR Flowchart

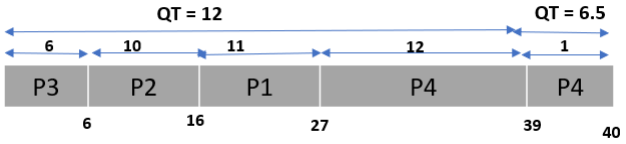


Figure 5: Same arrival time (IDRR) Gantt chart

$$Average\ waiting\ time\ (AWT) = \frac{49}{4} = 12.25$$

$$Average\ Turnaround\ time\ (ATT) = \frac{89}{4} = 22.25$$

Different Arrival Time (IDRR)

ID	Arrival Time (BT)	Burst time (BT)	Turnaround time	Waiting Time (WT)
P1	0	35	35-0 =35	0
P4	15	38	73-15 = 58	20
P3	9	50	168-9=159	109
P2	6	70	193-6=187	117

Table 4: Different arrival time (IDRR)

For the first iteration, P1 is the only task inside the ready queue. Hence, P1 is executed and the quantum time for 1st iteration = 35. During which, several other processes will be in the ready queue. QT will be updated as difference between average of highest remaining burst time + old QT, and with average of 2 lowest remaining arrival time. $\frac{35+70}{2} - \frac{6+9}{2} = 45$. With the updated QT, P4 will executed fully, while P3 will have a remaining of 5-unit time and P2 will have a remaining of 25. Again, quantum time will be updated with the remaining processes left in the ready queue. The highest remaining burst time is 25 from P2 and the lowest arrival time is 6. Hence the updated QT $= \frac{25+35}{2} - \frac{6}{2} = 27$ for the remaining processes, P2 and P4.

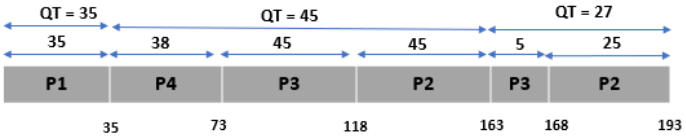


Figure 6: Different arrival time (IDRR) Gantt chart

$$Average\ waiting\ time\ (AWT) = \frac{246}{4} = 61.5$$

$$Average\ Turnaround\ time\ (ATT) = \frac{439}{4} = 109.75$$

Efficient Shortest Remaining Time Round Robin (ESRR):

Paper [4] describes a variation of the well-known RR combined with the SRTF algorithm to reduce waiting and turnaround time when compared to RR.

The proposed algorithm, called Efficient Shortest Remaining Time Round Robin (ESRR), uses both RR and SRTF (preemptive version of SJF). It uses a ready queue which stores all the processes arriving in a chronological order. The time quantum is chosen randomly.

The algorithm started off by considering the inputs as the number of processes, execution time and arrival time of all the processes, and allocates the CPU to the frontmost process. Now, if there are processes already in the ready queue and if the remaining execution time of the process is less than or equal to the assigned QT, the process will be allocated CPU and then removed from the ready queue once it finishes its execution. If the process does not finish its execution, it is preempted and put back in the ready queue for the next round of execution.

If a new process arrives and there are no existing processes in the ready queue, it is immediately allocated CPU for execution. Else, the execution time of the newly arriving process is compared to the existing processes in the ready queue from front to rear. If the execution time is less than all the existing

ones in the ready queue, it is put at the front of the ready queue. Otherwise, it is inserted at the end.

Same Arrival Time (ESRR)

ID	Arrival Time (BT)	Burst time (BT)	Turnaround Time (TT)	Waiting time (WT)
P1	0	11	33	22
P2	0	10	28	18
P3	0	6	6	0
P4	0	13	40	27

Table 5: Same arrival time (ESRR)

Consider the example as shown above. Let the time quantum be 6.

At t=0, all the processes are available in the ready queue. So, the process with least burst time is assigned CPU, which is P3, for time quantum. Then, it checks the next process with the least burst time which is P2. Now, P2 is assigned CPU for time quantum. After this, P1 and P4 are assigned CPU. Finally, P2, P1 and P4 come back for second round of execution.

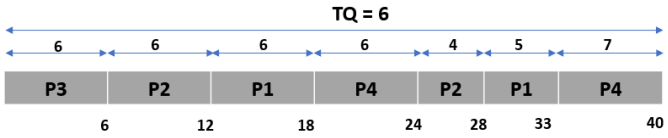


Figure 7: Same arrival time (ESRR) Gantt chart

$Average\ waiting\ time\ (AWT) = 16.75$

$Average\ Turnaround\ time\ (ATT) = 26.75$

Different Arrival Time (ESRR)

ID	Arrival Time (BT)	Burst time (BT)	Turnaround Time (TT)	Waiting time (WT)
P1	0	35	35	0
P2	6	70	177	107
P3	9	50	154	104
P4	15	38	58	20

Table 6: Different arrival time (ESRR)

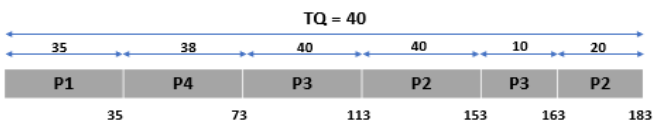


Figure 8: Different arrival time (ESRR) Gantt chart

Consider the example as shown above. Let the time quantum be 40.

First, at t=0, P1 is available in the ready queue which is assigned CPU for execution. At t=6, P2 arrives with burst time 70, but it is not assigned CPU as the remaining execution time for P1 is 29 which is less than P2's burst time. Again, at t=9 and t=15 P3 and P4 arrive, both of which are not assigned CPU as the remaining execution time for P1 is 26 at t=9 and 20 at t=15 respectively.

At t=35 P1 finishes execution, and P4 is assigned CPU as it has the least burst time among all the processes which finishes execution at t=73 (burst time = 38). Then P3 is assigned the CPU for time quantum and preempted after the time quantum.

P2 then takes up execution for time quantum, after which P3 comes back for the next round of execution along with P2. Refer to Appendix for the flowchart

$Average\ waiting\ time\ (AWT) = 57.75$

$Average\ Turnaround\ time\ (ATT) = 106$

Mean Difference Round Robin (MDRR):

Paper [3] proposes a preemptive algorithm which uses both variable and average time quantum for scheduling tasks. The proposed approach not only deals with average waiting time/ turnaround time but also, the problem of *starvation* which is prevalent in many preemptive based CPU scheduling algorithms.

It is noted that some algorithms designed to handle large waiting time and turnaround time fail to address the problem of starvation. A prime example of this is the SJF algorithm. SJF algorithm provides a way to reduced waiting time and turnaround time but can lead to starvation if there is a large burst time process in the queue with many short burst time processes. The proposed algorithm called MDRR tries to address this with the help of variable and average time quantum.

MDRR uses two queues namely, sorted ready queue and a waiting queue. It is assumed that the shortest process will be assigned to the CPU first. If a process completes its execution within the given time quantum, CPU is relinquished voluntarily. Otherwise, a partially executed process will be preempted and removed to be put into the tail of the ready queue for the next round of execution. If a process is waiting for input/output (I/O) during

execution, then it is immediately preempted and put into the waiting queue which can enter the ready queue later. It is also assumed that no new processes are added at the end of the ready queue.

MDRR first calculates the Average Time Quantum (TQ) by averaging the burst times across all the processes. Then, the Mean Difference (C) is calculated as sum of consecutive differences of sorted burst times divided by the total number of terms in the Mean Difference (C). Finally, the Variable Time Quantum (VTQ) is computed as Average Time Quantum added to the product of index (i) with Mean difference.

$$TQ = \frac{\sum_{i=1}^{N} P_i}{N} \qquad (4)$$

$$VTQ = TQ + i \times C \qquad (5)$$

Where,

i = Index of the process in the ready queue with Burst Time larger than TQ

C = Mean Difference

P_i *= Burst time of Process i*

N = Total Number of Processes

The Scheduling Algorithm first arranges all the processes in increasing order of their burst time. Then it calculates Mean Difference of all burst times of all the processes. Now, the Variable Time Quantum of each process is computed. If two or more processes have equal burst time, the CPU is assigned on FCFS basis. If burst time of a process is less than the average TQ then it will be executed within the TQ. Otherwise, the process will be executed for VTQ and added to tail of the sorted queue for the next round of execution if it is not finished within the given VTQ. Since this paper assumes that there are no new processes arriving; the following example considers only same arrival time.

Same Arrival Time (MDRR)

Consider the below-mentioned example. After sorting the processes based on their Burst Times, we have P3, P2, P1 and P4 in that particular order. The Average Time Quantum (TQ) is computed to be:

TQ = 10

ID	Arrival Time (BT)	Burst time (BT)	Turnaround Time	Waiting time
P1	0	11	27	16
P2	0	10	16	6
P3	0	6	6	0
P4	0	13	40	27

Table 7: Same arrival time (MDRR)

Processes P3 and P2 are assigned TQ as their burst times are less than or equal to TQ whereas, processes P1 and P4 have Burst Times greater than TQ. Hence, they will be assigned a Variable Time Quantum (VTQ). P1 is assigned the index i=1 and P4 is assigned the index i=2 as they are in that order in the sorted queue.

Mean Difference (C) is calculated by taking the sum of pairwise differences between the burst times in the sorted queue divided by the number of terms in the Mean Difference (here 3):

C = ((10-6) + (11-10) + (13-11)) / 3 = 2.33

Now,

VTQ_1 = 10 + 1*(2.33) = 12.33 (for P1)

VTQ_2 = 10 + 2*(2.33) = 14.66 (for P4)

Processes P3 and P2 will finish execution within the given time quantum i.e., TQ, whereas Processes P1 and P4 are assigned CPU for VTQ_1 and VTQ_2 respectively, so they complete execution within these time quanta.

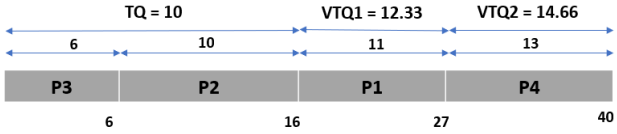


Figure 9: Same arrival time (MDRR) Gantt chart

$Average\ waiting\ time\ (AWT) = 12.25$

$Average\ Turnaround\ time\ (ATT) = 22.25$

Analysis & Comparison

In practice, Round Robin scheduling algorithm uses a fixed time quantum as compared to the proposed algorithms mentioned in this paper. Some of the proposed algorithms tackle the issues on starvation by ensuring that larger burst time processes are still given a chance to execute. This will drastically reduce the AWT and ATT.

Table 8 shows the AWT and ATT for both same and different tasks arrival based on different proposed algorithms.

	Same Arrival Time		Different Arrival Time	
	AWT	ATT	AWT	ATT
EDRR	13.25	23.25	53.25	101.50
IDRR	12.25	22.25	61.50	109.75
ESRR	16.75	26.75	57.75	106
MDRR	12.25	22.25	-	-

Table 8: AWT & ATT for all proposed algorithms

Based on the table shown above, for task arriving at the same time, IDRR and MDRR performs the best for AWT and ATT. This is because, IDRR ensures that larger tasks are given a chance to operate for the QT used. This prevents starvation from occurring. MDRR on the other hand, happens to have the same AWT & ATT as IDRR for the dataset above. It also addresses the problem of starvation by using mean difference time quantum.

Even though EDRR algorithm uses a dynamic Quantum Time, it is not necessarily more efficient than basic RR. It does not consider starvation and always allocates the smaller task to the CPU first making larger tasks wait in the ready queue. This can be seen on the Gantt chart in Figure 2. Hence, this causes the AWT to be longer as compared to IDRR and MDRR.

Although ESRR uses a dynamic time quantum to address the starvation issue, the algorithm is not that efficient as compared to the rest. This is mainly due to multiple context switches by incorporating SRTF into the algorithm. In addition, ESRR does not provide a criterion for choosing the quantum time, it is chosen randomly which is in stark contrast with rest of the algorithms.

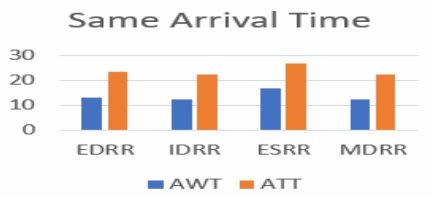


Figure 10: AWT & ATT Same Arrival Time

In the scenario of tasks arriving at different timing, Table 8 shows that EDRR has the best AWT, and ATT as compared to IDRR and ESRR. This is because EDRR ensures that the smaller tasks are always completed first even if larger task arrives. Similarly, ESRR follows this suit. This can be seen that the AWT & ATT is lower than IDRR.

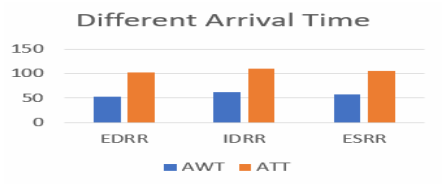


Figure 11: AWT & ATT Different Arrival Time

IDRR on the other hand tries to handle the issue on starvation. Therefore, in doing so, it will affect the AWT, and ATT based on this data set.

This is contradicting because, algorithms that handles starvation should have a lower AWT. This phenomenon can be attributed to the data set used.

Conclusion

This paper discussed 4 different variants of Round Robin scheduling algorithms – EDRR, IDRR, ESRR and MDRR compared on various aspects such as AWT, ATT, starvation, time quantum and arrival time.

Out of the 4 algorithms, IDRR, ESRR, and MDRR tackles the issue of starvation whereas EDRR neglects it.

The author of EDRR claims that EDRR is the most efficient scheduling algorithm with lowest AWT and ATT. This claim is invalid because EDRR does not address the problem on starvation and when compared to IDRR and MDRR, which considers starvation, the AWT and ATT is significantly lower.

The main disadvantage of MDRR is that it never considers the case of different arrival time. In addition, ESRR never provides a criterion for choosing the time quantum which makes the algorithm not stable across similar dataset.

In summary, based on the simulated dataset, IDRR and MDRR has a better AWT & ATT. However, for different arrival time, EDRR is the best. These observations should be taken with a grain of salt as the performance of the algorithms depends heavily on the task set used. A larger sample size datasets with a better distribution may provide a better evaluation of these algorithms.

References

[1] M. U. Farooq, A. Shakoor and A. B. Siddique, "An Efficient Dynamic Round Robin algorithm for CPU scheduling," *2017 International Conference on Communication, Computing and Digital Systems (C-CODE)*, 2017, pp. 244-248, doi: 10.1109/C-CODE.2017.7918936.

[2] A. Alsheikhy, R. Ammar and R. Elfouly, "An improved dynamic Round Robin scheduling algorithm based on a variant quantum time," *2015 11th International Computer Engineering Conference (ICENCO)*, 2015, pp. 98-104, doi: 10.1109/ICENCO.2015.7416332.

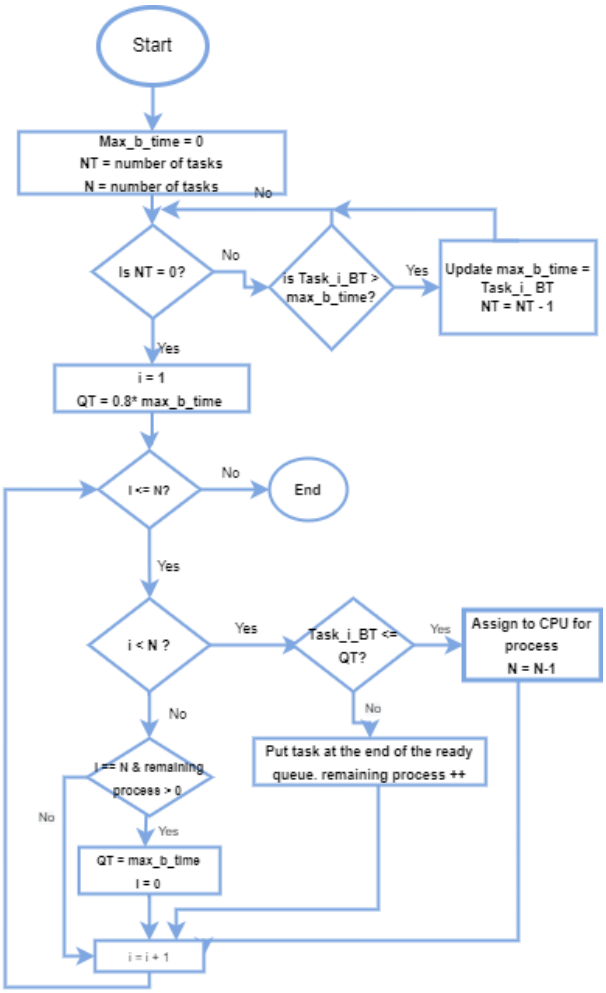
[3] S. Kumar Saroj, A. K. Sharma and S. K. Chauhan, "A novel CPU scheduling with variable time quantum based on mean difference of burst time," *2016 International Conference on Computing, Communication and Automation (ICCCA)*, 2016, pp. 1342-1347, doi: 10.1109/CCAA.2016.7813986.

[4] P. Sinha, B. Prabadevi, S. Dutta, N. Deepa and N. Kumari, "Efficient Process Scheduling Algorithm using RR and SRTF," *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, 2020, pp. 1-6, doi: 10.1109/ic-ETITE47903.2020.101.

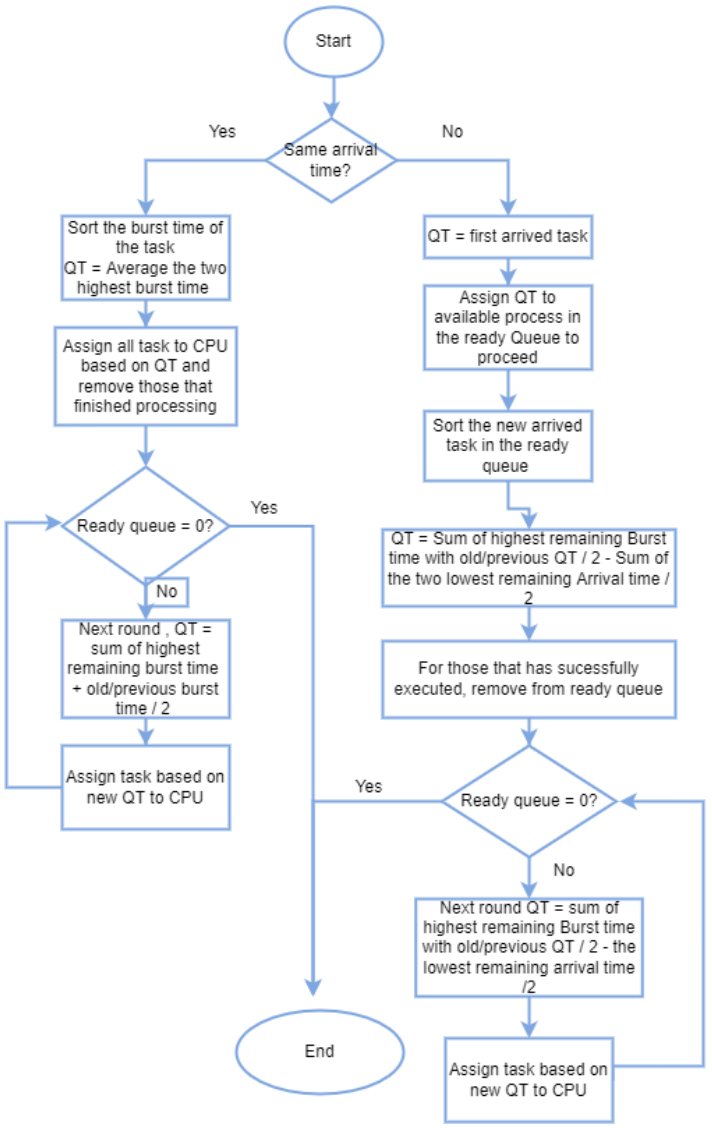
[5] V. Gondaliya, S. Patel, J. Hemnani and S. Patel, "Efficient vaccine scheduler based on CPU scheduling algorithms," *2021 International Conference on Artificial Intelligence and Machine Vision (AIMV)*, 2021, pp. 1-6, doi: 10.1109/AIMV53313.2021.9670986.

Appendix

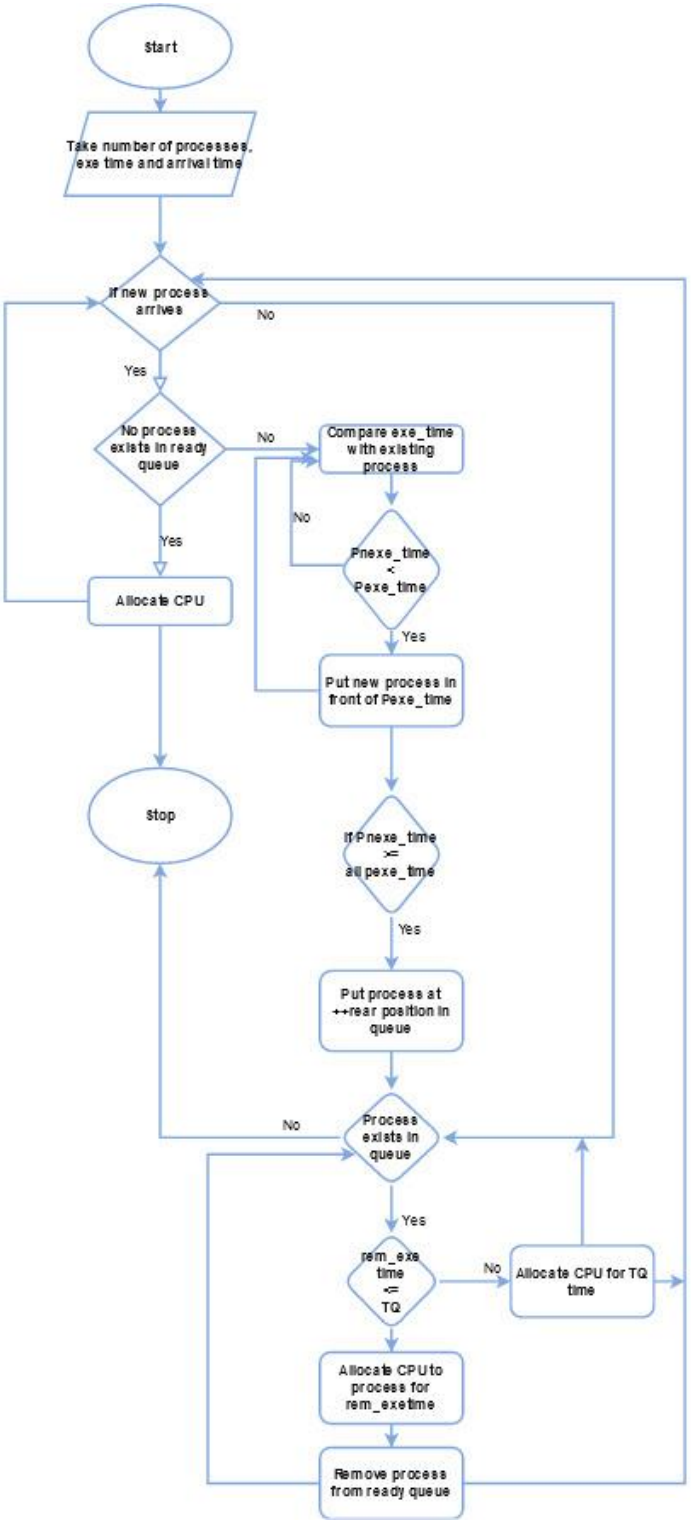
EDRR Algorithm



IDRR Algorithm



ESRR Algorithm



MDRR Algorithm