Efficient RR and SJF with Varying Time Quantum

1st Joel Toh
School of Infocomn Technology
Singapore Institute of Technology
Singapore, Singapore
2003288@sit.singaporetech.edu.sg

2nd Joash Lee Mau An
School of Infocomn Technology
Singapore Institute of Technology
Singapore, Singapore
2003286@sit.singaporetech.edu.sg

3rd Jarryl Wee
School of Infocomn Technology
Singapore Institute of Technology
Singapore, Singapore
2003278@sit.singaporetech.edu.sg

4th Chiu Yeow Keng
School of Infocomn Technology
Singapore Institute of Technology
Singapore, Singapore
2003274@sit.singaporetech.edu.sg

***Abstract*—Efficient process scheduling is crucial to multiprogramming operating systems. An inefficient process scheduling algorithm will very likely become a bottleneck, limiting speed and efficiency of the operating system. Common algorithms like shortest job first (SJF) and round robin (RR) are effective in their own ways, however they both have their fair share of downsides which will be further elaborated in our research. As such, we have come out with a more effective solution. Combining the concepts of 2 algorithms would negate their weaknesses, creating an all-rounded algorithm with better speed compared to the traditional round robin approach and at the same time, prevent starvation.**

Keywords—Process Scheduling, First Come First Serve, Shortest Job First, Round Robin, Turnaround Time, Waiting Time, Time Quantum.

I. Introduction

A. *Multiprocessing*

Multiprocessing is the utilization of two or more central processing units (CPU) within a single computer system. Asymmetric Multiprocessing and Symmetric Multiprocessing are the two variations of multiprocessing.

Asymmetric Multiprocessing - It is a multiprocessor computer system whereby the multiple interconnected CPUs are not treated equally. In asymmetric multiprocessing, a master processor will run the task of the operating system.

Symmetric Multiprocessing - It comprises of a multiprocessor computer hardware and software architecture whereby two or more identical processors will be connected to a single shared main memory, which will grant them full access to all input and output devices. Essentially, symmetric multiprocessing allows each processor to be self scheduling. [1]

B. *Scheduling*

Every process requires resources to accomplish its task. This is when CPU scheduling comes in handy. With multiprocessors systems growing in use and importance, scheduling has never been a more critical fundamental function of the operating system than now. Without proper scheduling algorithms, the operating system can be rendered useless, inefficient, slow and perhaps even crash. This is especially so in a multiprogramming operating system(OS), where CPU scheduling needs to be efficient. In a multiprogramming OS processes are switched in and out of the CPU while trying to ensure that CPU utilization is maximized.

To ensure that the CPU utilization is maximized, whenever the CPU is in idle mode, the OS minimally selects a process that is available in the ready queue for execution. Selection process will be done by the CPU scheduler and a ready process in the memory will be executed.

There are predominantly two types of scheduling methods, the preemptive and non-preemptive scheduling.

Preemptive Scheduling - For preemptive scheduling, tasks are usually assigned based on their priorities. In some instances, it is crucial to run a task with higher priority first, then another with a lower priority. This is done in cases where even in the instance where the lower priority task is still being run. The lower priority would be placed in the waiting queue and will resume after the higher priority task is finished with its execution. Some examples of Preemptive Scheduling are Round Robin (RR), Shortest Remaining Time First (SRTF)

Non-Preemptive Scheduling - For non-preemptive scheduling, the CPU will be allocated to a specific process. This process will be run by the CPU and will only release the CPU when switching context or termination. On some hardware platforms, this is the only method that can be employed. This is because such scheduling methods do not require special hardware, like timers, unlike preemptive scheduling. Some examples of non-preemptive scheduling are, First Come First Serve (FCFS), Shortest Job First (SJF) and Priority Scheduling. [2]

C. *Brief summary of known algorithm*

Algorithms – a set of instructions used to solve a problem. In computing, algorithms are the keys to provide computers the steps to execute a task. There is no go-to algorithm that exists, each algorithm has their own advantages and disadvantages that are dependent on its deployment.

There are four popular methods of scheduling algorithms, namely First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR) and Priority-based Scheduling.

- First Come First Serve, a simple, straightforward algorithm that is easy to implement where it “serves” the first process request, following the next in queue, so on and forth. This ensures that all processes in the queue will run. As there is no process priority, processes that are ready to run will not be waiting indefinitely (starvation). However, the algorithm does not support pre-emption (where running higher priority processes take precedence), processes that are placed at the end of the queue may be bottlenecked if processes prior to it have a long burst time.

- Shortest Job First as the name suggests, allocates resources to run the shortest processes first, followed by the longer processes. In return, the throughput (the performance of the computer system over a specific period of time) of the computer system will be increased, as more processes are being executed. On the flipside, as the shortest processes get to be completed first, it will cause a longer waiting time for the longer processes, which may eventually lead to starvation.

Processes in the Round Robin Scheduling Algorithm are given the same priority with a fixed time quantum. The process runs in a round robin cycle, where each process is given a fixed time to execute based on the quantum. However, as processes are not given any special priority, important tasks do not get priority over other tasks. The throughput of the algorithm is also largely based on the quantum time, where if it were to be set too low, it will reduce CPU efficiency and possibly cascading of processes due to the constant switching of it.

In Priority-based Scheduling, processes can be given a priority based on its memory requirement, time requirement or user preference. However, when there are several processes of the same priority, a second algorithm is required to schedule the process.[3]

D. *Performance-based selection criteria*

There are many CPU scheduling algorithms of various techniques. How an operating system performs is mainly dependent on how the algorithm functions. The criteria to select and improve on the choice of preferred scheduling algorithm is dependent on the following parameters.

- CPU utilization [4]: It is an average measurement of the percentage of how busy the CPU is. 100% being the max and 0% being the lowest. The higher the CPU utilization rate, the more effective the OS will be. Best case scenario would be to constantly have a high CPU utilization rate as this would mean that the CPU is making use of every one of its cycles.

- Burst time [4]: It is defined as the period of time which a process is being run by the CPU until its completion.

- Time Slice: Is defined as the maximum duration of time a process can own the CPU for. After which another process will be allocated to the CPU and the previous process will join the waiting queue. This is to ensure fairness and prevent starvation. This is only used in certain scheduling algorithms.

- Waiting time [4]: It is defined as the total time in which a process waits for the CPU in the queue. This criterion should be kept to a minimum to enhance performance.

- Turnaround Time [4]: It is defined as the total time the process takes from arrival to completion. This criterion should be kept to a minimum to enhance performance.

II. Background Research and Relevant Research Papers

A. *Round Robin*

Round Robin(RR) is one of the common algorithms employed by schedulers in computing. Time slices are assigned to each process in equal portions and in a circular order, regardless of priority.

The downside of this algorithm is that setting the time quantum too short would increase the overhead and lower the CPU efficiency but setting the quantum too long would cause poor responses to short processes. Another disadvantage is that processes would not have any priority, which means that high priority processes would still have to wait for an equal amount of time compared to the other processes instead of waiting less due to its high priority nature.

In paper [4], Sorted Round Robin Algorithm is one of the methods for process scheduling, a key feature of a Multiprogramming operating system where the process manager deals with the choice and elimination of various processes. The algorithm combines the advantages of both the shortest job first (SJF) and Round Robin (RR) algorithm while improving on the shortcomings of the RR algorithm.

In order to maximize the utilization of the CPU, the CPU is alternated between various processes. The RR algorithm is framed mainly for a time-sharing system. “The mechanism being that each process is executed in First Come First Serve (FCFS) order in the given time slice and those greater than that time quantum are sent to the back of the ready queue, where the remaining processes are waiting for their execution. The selection of time slice is important as it affects the performance of algorithm”

In paper [5], the author starts off by talking about the existence of many Scheduling Algorithms such as First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), Priority Scheduling, etc. In order to reduce the time a process spends in a waiting state and avoid starvation of processes, a strategy consolidated of three existing scheduling algorithms(RR, SJF, Priority) is created.

The algorithm works by first assigning all the processes in the job queue to have a priority, which ranges from low, medium to high. The processes with low priority would be given a smaller time slice compared to the medium priority processes and vice versa with the medium priority processes and high priority processes. Improved Round Robin algorithm uses the priority of the processes to assign the time quantum of each priority statically. The throughput of the algorithm is increased by executing processes with smaller burst time or smaller remaining burst time by allocating the process to the CPU as soon as it is ready but not preempting the running process.

When compared to the Round Robin algorithm, overall the Improved Round Robin algorithm had a lower average turnaround time, lower number of context switches and a lower average waiting time. By introducing priority in the RR algorithm, the time quantum can be decided separately for each process and the CPU can be shared in a fairer way than in the existing algorithm. This algorithm solves the aforementioned disadvantage brought by RR that was having no priority by incorporating priority scheduling.

In paper [6], the primary objective of the algorithm is to build up another methodology for round robin CPU booking calculation in which improving the presentation of CPU progressively working framework. The algorithm depends on the incorporation of Round Robin(RR) and Need Booking calculation. This way, the algorithm would hold the upside of RR scheduling in reducing starvation and incorporating the benefit of need booking.

The algorithm would first allocate all the process towards the ready queue. It would then allocate CPU to every process in a RR fashion, according to the given priority, for a given time quantum only once. After that, the processes are arranged in increasing order or remaining CPU burst time in the ready queue. New priorities are then assigned according to the remaining CPU bursts of the processes. After calculating the new dynamic time quantum using a given formula, the processes would run according to their new priorities and new dynamic time quantum and the processes of arranging and calculation of dynamic time quantum would repeat till the ready queue is empty.

This new algorithm sees an improvement compared to the existing Round Robin algorithm in terms of lower average waiting time and lower average turnaround time. The algorithm lessens the working framework overhead and lessens the issue of starvation as the procedures with less outstanding CPU burst time are allocated with the higher needs and would be executed first. This algorithm solves the aforementioned disadvantage brought by RR that was having no priority and low CPU efficiency due to time quantum value by incorporating a priority allocation and a calculation for the time quantum value.

B. Shortest Job First

Shortest Job First(SJF) is a non-preemptive scheduling algorithm that selects the waiting process with the least amount of execution time for execution. The algorithm is advantageous due to its simplicity and being able to minimize the average amount of time each process has to wait until its execution is done. However, with the SJF scheduling algorithm, there will be a possibility of starvation. This will occur if shorter processes constantly arrive, leading to longer processes not being able to run. [7]

In paper [8], it highlights the flaws that the SJF algorithm downside is that it has the potential to cause process starvation. It is especially so for processes that require a longer time to complete, should there be a continuing addition of short processes. Also, the total execution time of a job must be known before execution.

Scheduling algorithms work differently based on the different kinds of variables like their time-stamp value, service time, waiting time, priority as well as response ratio. In Shortest Job First(SJF), the process with the least CPU burst time will be allocated the CPU first and in Round Robin(RR), a time quantum would be given to each process. The Shortest Job Round Robin(SJRR) Algorithm is based on the combined approach of RR and SJF with a threshold value.

In this algorithm, the threshold value will be the mean of all the process's burst time. The burst time of each process is then compared to the threshold value. The processes whose burst time is less than or equal to the threshold value are placed in a separate queue and SJF would be applied in the queue of processes. The processes whose burst time are greater than the threshold value are placed in another queue and have RR applied on them.

This algorithm sees improvement in the wastage of time in context switching and the job fairness. The CPU is kept in the idle state for the least time compared to RR and its CPU utilization is also higher. It is also leading SJF in starvation, as none of the jobs have to wait a long time. This algorithm solves the previously mentioned downside of SJF which is process starvation by sorting out the processes by burst time then allocating them to their respective queues.

III. Proposed Algorithm

The proposed algorithm design is designed to negate the downsides of the basic round robin algorithm and at the same time empower it with the most efficient Shortest Job First algorithm. The basic round robin algorithm is created to give the same need to every single one of the processes. This downside of the basic round robin engineering causes it to become inefficient should processes be run with a small time slice. This will generally result in the CPU doing many contact switches. On the contrary, should processes be run with a big time slice, waiting time and turnaround time would increase.

The proposed algorithm is an improved round robin algorithm, which incorporates a varying factor "x" carefully calculated based on statistics to find the best varying time slice value. Throughout the scheduling process, it would be arranging the process in the ready queue based on which process has the shorter job and put it first.

The proposed algorithm will tackle 2 main issues:

- Starvation - In some algorithms, such as Shortest Job First, starvation will occur such that longer processes will not be able to run due to the constant arrival of shorter processes. As such by using round robin as the base for the proposed algorithm, starvation will be prevented as expounded upon in the background research portion of the report.
- Reduce waiting and turnaround time - As mentioned in the research portion of the report, Round Robin algorithm's efficiency is heavily based on how the value of the time slice is calculated. The proposed algorithm utilizes a formula, tested with statistics, to calculate a varying round robin time slice value. This value will change every round based on the remaining time of the processes in queue. This way optimal efficiency will be maintained. Furthermore, combining Round Robin with the Shortest Job First algorithm makes the selected algorithm even faster. When running the shorter process in the queue first, in a round robin fashion, the turnaround and waiting time will be further reduced.

A. Pseudocode and Flowchart

```
1. Initialize empty Process Array
2. Open file for reading, and add each line as a Process to the Process Array
3. Initialize TQ as Average Burst Time of Process Array / 2
```

```
4. Sort Processes in Array based on Arrival Time, and then by Burst Time
5. For each Process in Process Array
    - Check if PC == total_processes, if so, reset PC to 0
    - Assign Pointers for current, next and previous process (current_proc, next_proc, prev_proc)
            - If prev_proc has finished running AND End Time for prev_proc is earlier than Arrival Time of current_proc AND Run Count of current_proc is 0,
            - Set time_counter to be Arrival Time of current_proc
    - Check if Burst Time of current_proc is 0, move to next Process if so
    - Else, run Process
    - If Burst Time of current_proc is more than TQ,
            - Run Process for TQ duration
                    - Add TQ duration to time_counter
                    - Subtract TQ duration from Burst Time of current_proc
            - Check if Remaining Burst Time of current_proc is less than 1/2 of TQ, run Process fully if so
            - Add Remaining Burst Time of current_proc to time_counter
            - Set Burst Time of current_proc to be 0
    - Set End Time of current_proc to be the same as time_counter
    - Check if Burst Time of current_proc is 0, if so:
        - Calculate Waiting and Turnaround Time for current_proc
    - Increase Run Count of current_proc
    - Check if End Time of current_proc is earlier than Arrival Time of next_proc
            - Go to first unfinished Process in the Array
    - Check if all Processes have finished running, exit loop if so
6. Print out results
7. Clear memory used by Process Array
```

Flow chart as depicted in Figure.2.

B. Finding best multiplier for Time Quantum

In search of the best multiplier to the proposed algorithm, 5 different datasets were put to the test with varying Time Quantum (TQ) multipliers. The ranking was based on the shortest average waiting time and average turnaround time with 1st being the fastest and 5th being the slowest. The results were as such:

Table.1. Comparison to find best TQ multiplier

TQ multiplier	Data Set 1	Data Set 2	Data Set 3	Data Set 4	Data Set 5
0.25	5th	2nd	4th	4th	5th
0.50	3rd	5th	4th	3rd	4th
0.75	4th	**1st**	3rd	2nd	3rd
1.00	2nd	4th	2nd	1st	2nd
1.25	**1st**	3rd	**1st**	**1st**	**1st**

From the results, the obvious best TQ multiplier was 1.25. To see the exact values of the data sets, refer to Figure 3-7.

C. Example

Table.2. Process with sorted arrival time and burst time

Process	Arrival Time	Burst Time
P8	0	2
P6	2	4
P10	3	6
P9	4	2
P5	4	5
P1	7	5
P2	7	9
P3	9	6
P4	9	6
P7	9	10

The total burst time of the above example is 55 and the average burst time would be 5.5. Hence 1.25*5.5 will give you 6.875 which will be rounded down to 6 as the time slice.

However, after P2 runs the full 6 seconds of time slice, the algorithm will check its remaining burst time. If it is more than half the calculated time slice, in this case it will be 6 divided by 2 which is 3, it will move on to the next process. In this case, because the remaining time slice of P2 is 3 and 3 is not bigger than 3 (half the calculated time slice), the algorithm will run the remaining burst time of P2. Refer to Figure.1. for the gantt chart for the example above.

D. *Comparison*

A simple comparison between traditional Round Robin (RR), Shortest Job First (SJF), First Come First Serve (FCFS) and the proposed algorithm to see how it fairs.

The values will be based on the previous example in Table.2.

Table.3. Algorithm time comparison table

Algorithm	Average Waiting Time	Average Turnaround Time
RR (slowest time slice)	23.4	28.9
RR (fastest time slice)	14.3	19.8
FCFS	14.3	19.8
SJF	12.8	18.3
Proposed Algorithm	14	19.5

E. *Strengths and weakness*

The proposed algorithm strength comes from the ability to combine two common algorithms together and in doing so negate their weaknesses. It is able to prevent starvation by using the Round Robin algorithm as its base. It then sorts based on Shortest Job First, based on the time at which it arrives. This cuts down on the longer waiting time as shorter processes get to run and complete first. Furthermore, the proposed algorithm has a checking method which will run the remaining process burst time if it is lower or equal to half the varying time slice value. This is a key factor in helping to reduce its waiting and turnaround time even more.

The weaknesses would be if there were any overheads for Round Robin due to switching from a process to another. This would increase the average waiting time and turnaround time for the proposed algorithm.

IV. CONCLUSION

In this report, we have presented our Efficient RR and SJF with Varying Time Quantum algorithm. After a huge amount of research and testing, we managed to combine two of the common algorithms and yet overcome their weaknesses and even complement each other with their strengths. The results of the proposed algorithm have shown to have better performance compared to the traditional Round Robin algorithm and the First Come First Serve algorithm.

REFERENCES

[1] "Difference between asymmetric and symmetric multiprocessing," GeeksforGeeks, 25-Nov-2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-asymmetric-and-symmetric-multiprocessing/. [Accessed: 01-Apr-2022].

[2] L. Williams, "Preemptive vs non-preemptive scheduling: Key differences," Guru99, 24-Feb-2022. [Online]. Available: https://www.guru99.com/preemptive-vs-non-preemptive-scheduling.html. [Accessed: 01-Apr-2022].

[3] "Comparison of scheduling algorithms," Studytonight.com. [Online]. Available: https://www.studytonight.com/operating-system/comparision-scheduling-algorithms. [Accessed: 02-Apr-2022].

[4] R. Srujana, Y. Mohana Roopa, and M. Datta Sai Krishna Mohan, "Sorted Round Robin Algorithm" Institute of Aeronautical Engineering Hyderabad, India , Tech. Rep. 978-1-5386-9439-8, 2019.

[5] F. Alaa, M. Maaza Zoulikha and B. Hayat, "Improved Round Robin Scheduling Algorithm With Varying Time Quantum" IMPA LAB USTO-MB Oran, Algeria, Tech. Rep. 978-1-7281-7503-4, 2020.

[6] P. Chand Katoch1 and I. Sharma, "Modified Round Robin Scheduling Algorithm Based on Priorities" RMEC, Rajasthan Technical University Kota, Rajasthan, India, Tech. Rep. ISSN 2321 3361, 2019

[7] "Advantages and disadvantages of various CPU scheduling algorithms," GeeksforGeeks, 27-Jan-2022. [Online]. Available: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-various-cpu-scheduling-algorithms/. [Accessed: 02-Apr-2022].

[8] P. Pathak, P. Kumar, K. Dubey, P. Rajpoot, "Mean Threshold Shortest Job Round Robin CPU Scheduling Algorithm" REC Ambedkar Nagar, Akbarpur, India, Tech. Rep. 978-1-5386-7799-5, 2019

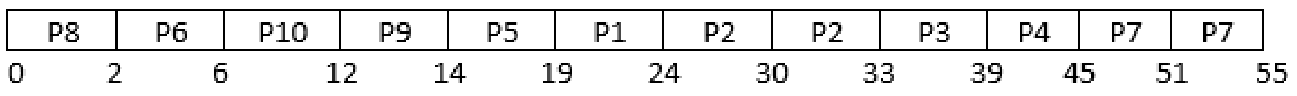
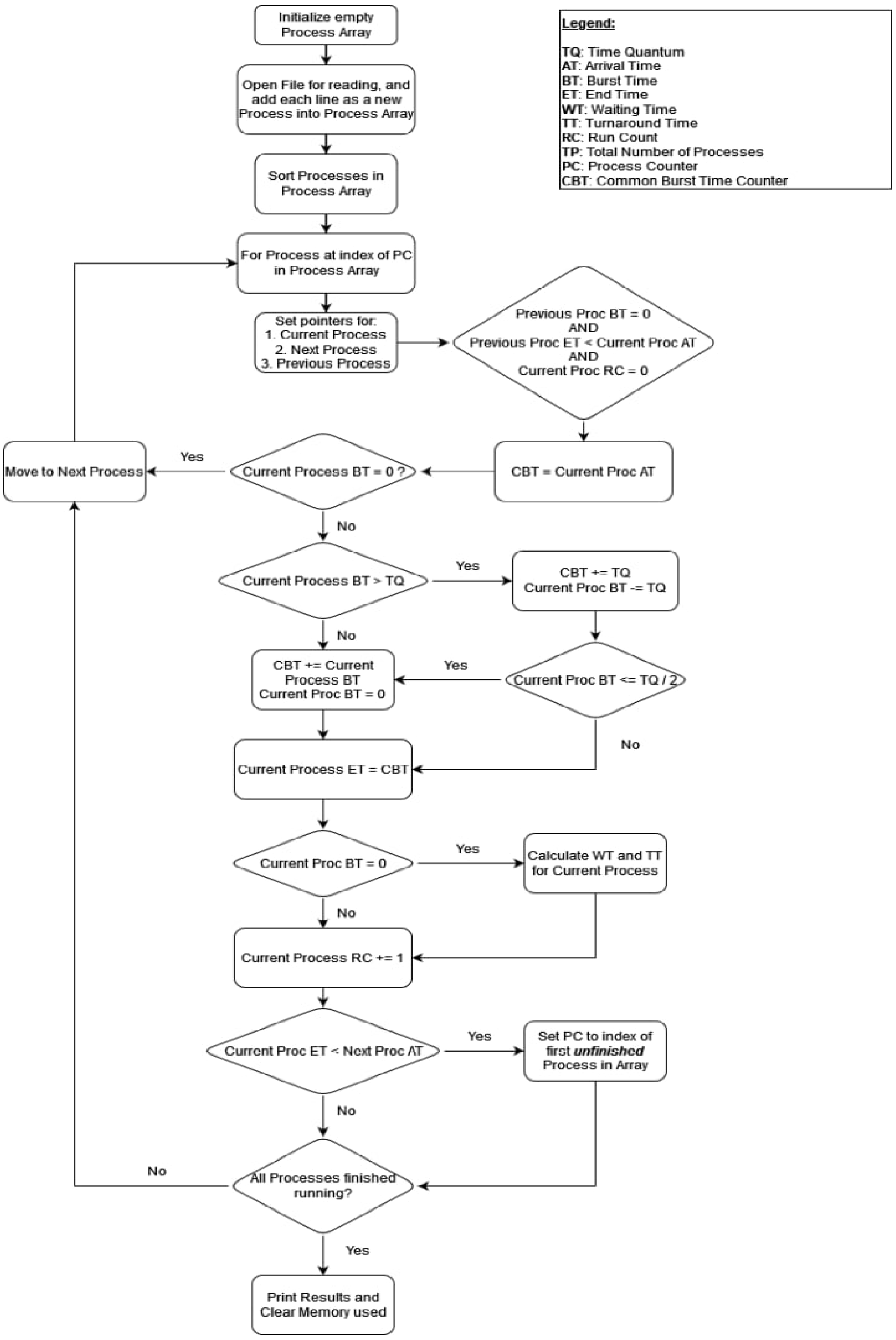
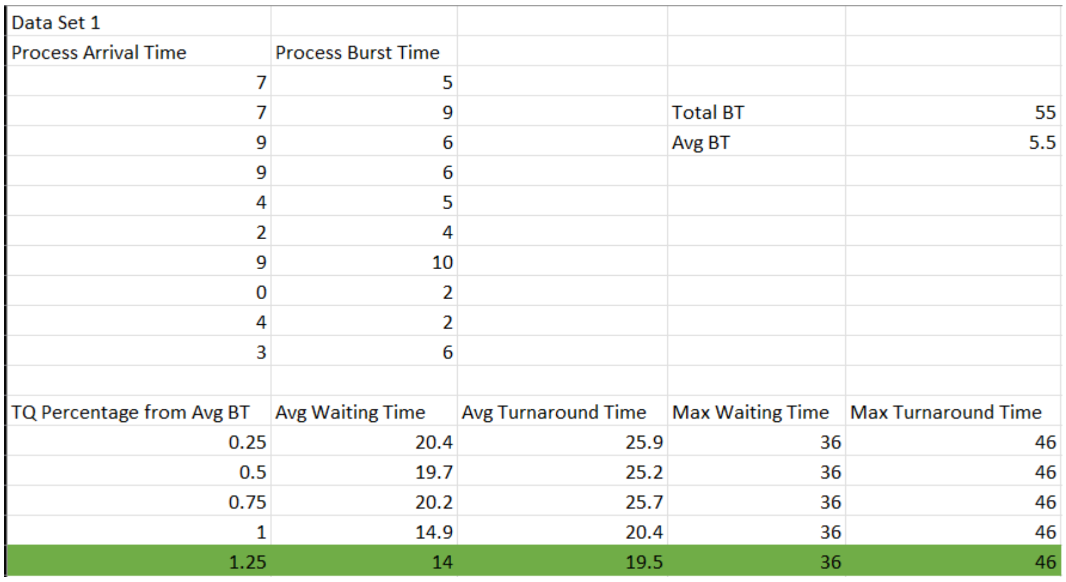


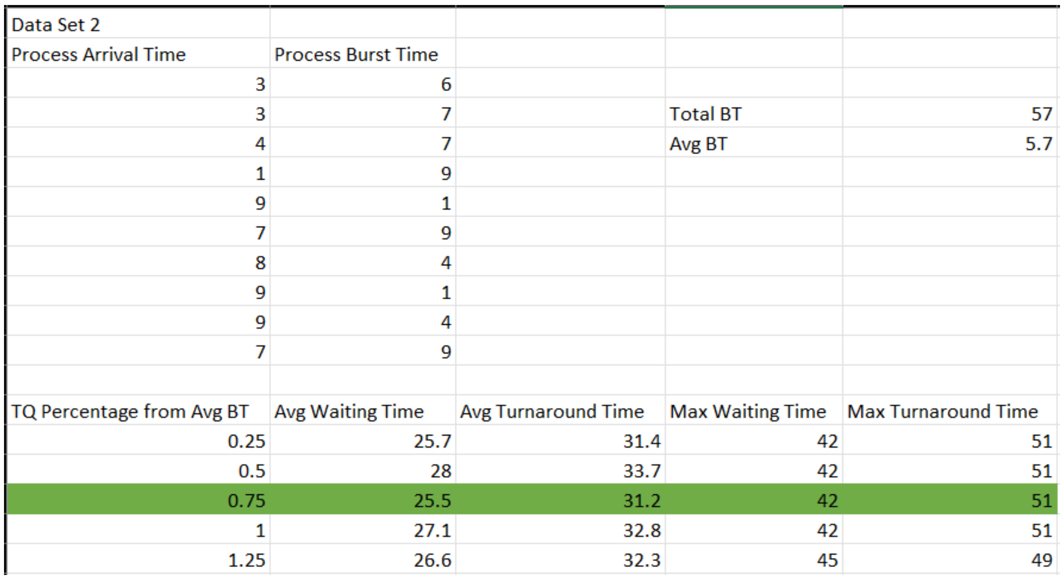
Figure.1. Gantt Chart Using Proposed Efficient RR and SJF with Varying Time Quantum Scheduling Algorithm


Legend:
TQ: Time Quantum
AT: Arrival Time
BT: Burst Time
ET: End Time
WT: Waiting Time
TT: Turnaround Time
RC: Run Count
TP: Total Number of Processes
PC: Process Counter
CBT: Common Burst Time Counter
Initialize empty Process Array
Open File for reading, and add each line as a new Process into Process Array
Sort Processes in Process Array
For Process at index of PC in Process Array
Set pointers for: 1. Current Process 2. Next Process 3. Previous Process
Previous Proc BT = 0 AND Previous Proc ET < Current Proc AT AND Current Proc RC = 0
CBT = Current Proc AT
Current Process BT = 0 ?
Yes
Move to Next Process
No
Current Process BT > TQ
Yes
CBT += TQ Current Proc BT -= TQ
No
Current Proc BT <= TQ / 2
Yes
CBT += Current Process BT Current Proc BT = 0
No
Current Process ET = CBT
Current Proc BT = 0
Yes
Calculate WT and TT for Current Process
No
Current Process RC += 1
Current Proc ET < Next Proc AT
Yes
Set PC to index of first unfinished Process in Array
No
All Processes finished running?
No
Yes
Print Results and Clear Memory used
Figure.2. Flow Chart



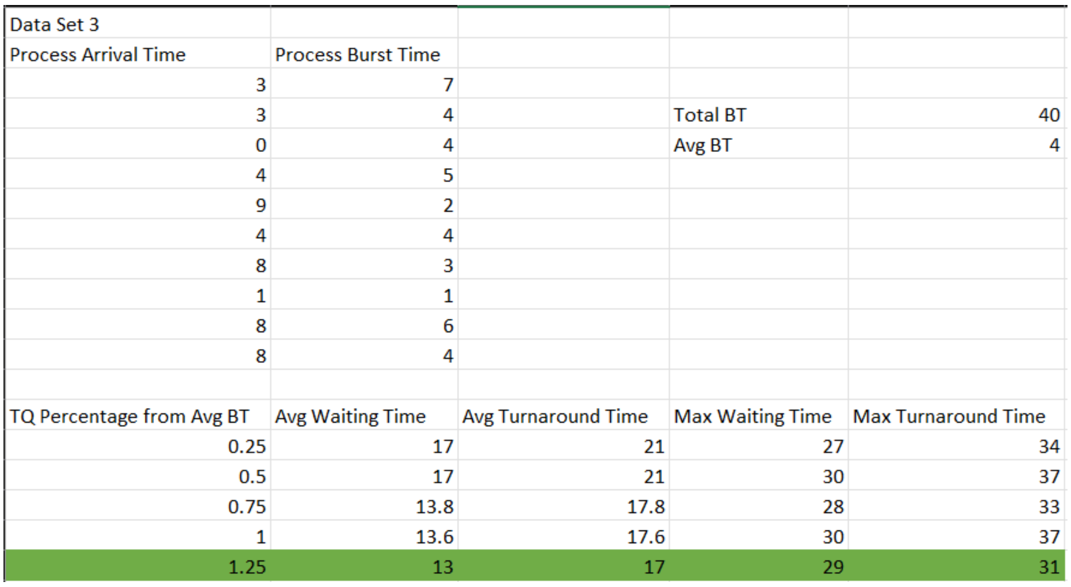
Data Set 1				
Process Arrival Time	Process Burst Time			
7	5			
7	9		Total BT	55
9	6		Avg BT	5.5
9	6			
4	5			
2	4			
9	10			
0	2			
4	2			
3	6			
TQ Percentage from Avg BT	Avg Waiting Time	Avg Turnaround Time	Max Waiting Time	Max Turnaround Time
0.25	20.4	25.9	36	46
0.5	19.7	25.2	36	46
0.75	20.2	25.7	36	46
1	14.9	20.4	36	46
1.25	14	19.5	36	46

Figure.3. Data Set 1



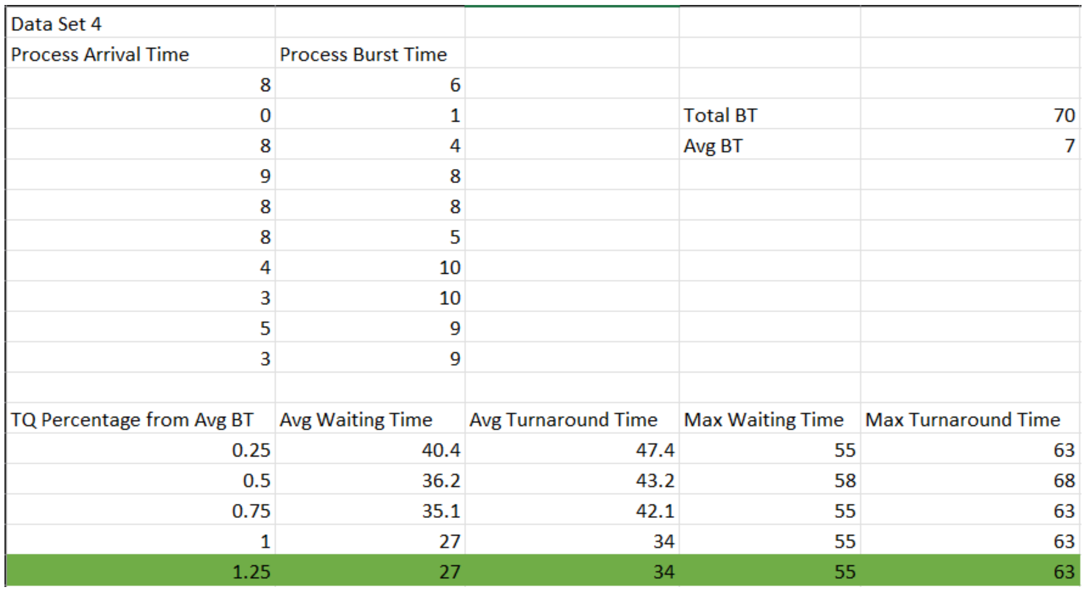
Data Set 2				
Process Arrival Time	Process Burst Time			
3	6			
3	7		Total BT	57
4	7		Avg BT	5.7
1	9			
9	1			
7	9			
8	4			
9	1			
9	4			
7	9			
TQ Percentage from Avg BT	Avg Waiting Time	Avg Turnaround Time	Max Waiting Time	Max Turnaround Time
0.25	25.7	31.4	42	51
0.5	28	33.7	42	51
0.75	25.5	31.2	42	51
1	27.1	32.8	42	51
1.25	26.6	32.3	45	49

Figure.4. Data Set 2



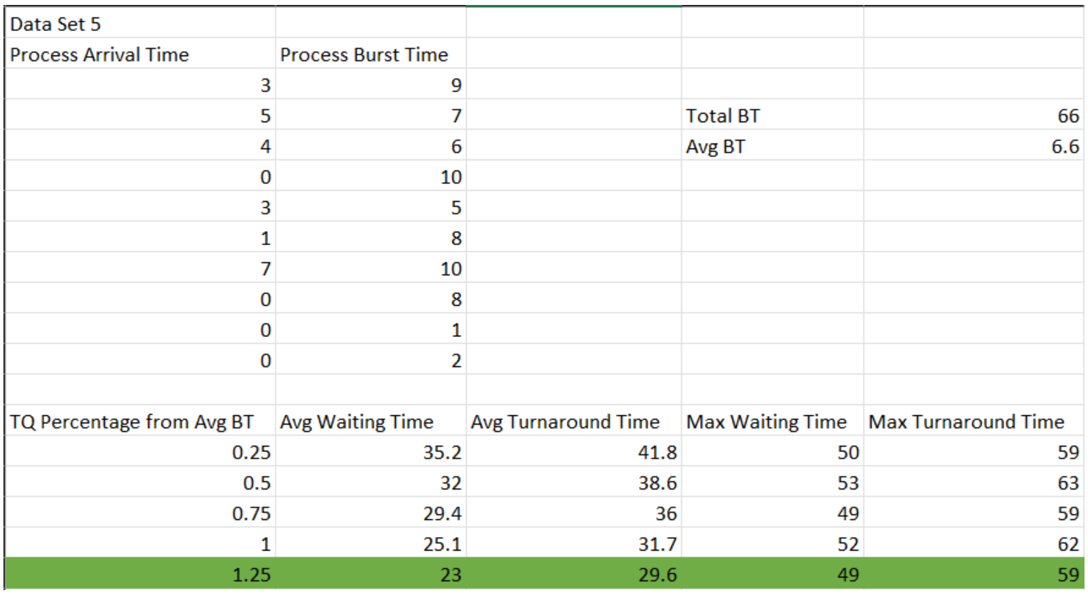
Data Set 3				
Process Arrival Time	Process Burst Time			
3	7			
3	4		Total BT	40
0	4		Avg BT	4
4	5			
9	2			
4	4			
8	3			
1	1			
8	6			
8	4			
TQ Percentage from Avg BT	Avg Waiting Time	Avg Turnaround Time	Max Waiting Time	Max Turnaround Time
0.25	17	21	27	34
0.5	17	21	30	37
0.75	13.8	17.8	28	33
1	13.6	17.6	30	37
1.25	13	17	29	31

Figure.5. Data Set 3



Data Set 4				
Process Arrival Time	Process Burst Time			
8	6			
0	1		Total BT	70
8	4		Avg BT	7
9	8			
8	8			
8	5			
4	10			
3	10			
5	9			
3	9			
TQ Percentage from Avg BT	Avg Waiting Time	Avg Turnaround Time	Max Waiting Time	Max Turnaround Time
0.25	40.4	47.4	55	63
0.5	36.2	43.2	58	68
0.75	35.1	42.1	55	63
1	27	34	55	63
1.25	27	34	55	63

Figure.6. Data Set 4



Data Set 5				
Process Arrival Time	Process Burst Time			
3	9			
5	7		Total BT	66
4	6		Avg BT	6.6
0	10			
3	5			
1	8			
7	10			
0	8			
0	1			
0	2			
TQ Percentage from Avg BT	Avg Waiting Time	Avg Turnaround Time	Max Waiting Time	Max Turnaround Time
0.25	35.2	41.8	50	59
0.5	32	38.6	53	63
0.75	29.4	36	49	59
1	25.1	31.7	52	62
1.25	23	29.6	49	59

Figure.7. Data Set 5