https://dx.doi.org/10.4314/dujopas.v8i3a.11
ISSN (Print): 2476-8316
ISSN (Online): 2635-3490
Dutse Journal of Pure and Applied Sciences (DUJOPAS), *Vol. 8 No. 3a September 2022*

Improved Shortest Job First CPU Scheduling Algorithm

Abraham Eseoghene Evwiekpaefe[1], Abdulrasheed Ibrahim[2] and Muhammad Nazeer Musa[3]

[1,2,3] Department of Computer Science,
Nigerian Defence Academy,
Kaduna, Nigeria

Email: aeevwiekpaefe@nda.edu.ng[1]

Abstract
CPU scheduling is one of the most important tasks of the Operating System (OS). Among the traditional scheduling technique, Shortest Job First (SJF) scheduling is an excellent choice for minimizing the average waiting time of a group of available processes. It is notable for allocating less average waiting time to available processes and more waiting time to processes that require more time to complete execution. When small processes arrive in large numbers on a regular basis, long processes become starved. This paper proposed an improved scheduling strategy to aid task scheduling while minimizing the starvation problem associated with the shortest Job First algorithm thus providing a better and more efficient waiting time for processes with longer burst times. The method took an innovative approach by first giving the CPU the shortest burst time and then using the average of the remaining sorted burst timings. The proposed technique was built and compared against other algorithms utilizing two different statistical distributions (namely Poisson and Binomial distributions). When compared with other improved algorithms like SRDQ, HYRR, and ADRR, the proposed technique achieved better Average Waiting Time and Turnaround time for processes with longer burst times in all statistical distributions while reducing starvation. The implementation was done using simulation for observing the behavior of the CPU scheduling algorithms and the simulator was written in java, specifically in NetBeans IDE 6.9.1.

Keywords: CPU scheduling, waiting time, turnaround time, starvation, IMPSJF, SJF

INTRODUCTION
An operating system (OS) is a piece of system software that acts as a bridge between the user and the computer hardware. The OS also provides a platform for users to interact with hardware and efficiently run programs (Silberschatz et al., 2018). Modern OS and time sharing systems are more sophisticated because they have progressed from a single job to a multitasking environment in which tasks execute in synchrony. If numerous processes are ready to execute at the same time in a multiprogrammed environment, the system must schedule them to run on the available Central Processing Unit (CPU).

CPU scheduling is the process of scheduling incoming processes to the CPU and this happens to be the basis for multiprogramming OS. As a result, keeping processes active at all times helps to maximize CPU efficiency, so that when the CPU switches between processes, the operating system makes the machine more productive. In a single processor system, only one process may run at a time; other programs must wait until the CPU is free before being

**Author for Correspondence*

rescheduled. As a result, the CPU remains idle. The waiting time is squandered during this period, and no valuable work is completed. Several processes are held in memory at the same time in multiprogramming. As a result, time is spent effectively (Silberschatz et al., 2018). When there are more processes in the ready queue waiting to be assigned to the CPU, the processes are made to wait for the OS to release the CPU from the currently running process and decide the order of execution of the other processes in the ready queue. The scheduler is the part of the operating system in charge of making this decision, and the algorithm it applies is known as the scheduling algorithm (Tanenbaum and Bos, 2015). The purpose of scheduling algorithms is to reduce process turnaround time, waiting time, average waiting time, and the frequency of context switches (Saroj and Roy, 2013). CPU scheduling algorithms include Priority scheduling, First-Come-First-Served, Shortest Job First (SJF), and Round Robin (RR). The approach in these existing algorithms can still conduct the same task, but in a simpler manner that requires fewer resources and less execution time to achieve a better result.

The Shortest Job First Scheduling method produced a better Average Waiting Time (AWT) and Average Turnaround Time (ATT), but the algorithm always starve the process with the longer burst time and made them wait for a long period until there were no other shorter processes, causing the longer process to remain stuck (Shafi et al., 2019).

Elmougy et al. (2017) conducted a research and developed a hybrid algorithm known as SJF and RR with dynamic quantum hybrid algorithm (SRDQ), the algorithm attempted to partially overcome the starvation problem of long tasks by proposing a hybrid scheduling algorithm based on two traditional scheduling algorithms SJF and RR. These two algorithms were purposefully chosen to take advantage of SJF fast scheduling while overcoming its starving problem with RR augmented with dynamic quantum. Experiment results and tests showed that the suggested approach outperformed the state of the art in terms of waiting time, response time, and partially starving of long tasks, but it also has computational overhead due to too many context shifts preempting often. Later, Shafi et al. (2019) used dynamic quantum time rather than fixed quantum time in his proposed Amended Dynamic Round Robin (ADRR). After the processes were organized in ascending order, the time quantum was also measured based on the process with the shortest burst duration. The difference between it and other hybrids (SJF and RR) is that it establishes a threshold value of Quantum Time (QT) and then tests a condition. This algorithm performs better if the processes have a longer burst period and arrive at the same moment, otherwise it has a significant number of context shifts when compared to SJF.

Ali et al. (2020) also proposed a new approach to scheduling with an enhanced time quantum based algorithm. This enhancement was by using dynamic time quantum leads to minimize AWT, ATT, ART and NOC. This approach inherited the properties of RR, SJF algorithm and FCFS algorithm. Therefore, the algorithm is a hybrid round robin scheduling mechanism for process management (HYRR Mechanism). HYRR Mechanism was innovative in that it which reduced ATT, AWT and Number of CS to the desired levels without starving longer processes.

From the reviewed works of Elmougy et al. (2017), Shafi et al. (2019) and Ali et al. (2020), it shows that they have certain limitations like low throughput and high computational overhead when compared to SJF. This is because all of these improved algorithms were pre-emptive, using either tiny or massive quantum time, and still did not outperform SJF, since it is optimal (Teraiya & Shah, 2018). The algorithm proposed in this research is an extension of non-preemptive SJF that reduces starvation of processes with larger burst times while having

a low computational overhead and a decent AWT and ATT equivalent to SJF but better than other modified algorithms, as well as having fewer context shifts than prior modifications.

METHODOLOGY

The Proposed Algorithm

The proposed algorithm assumes that all processes are in the READY QUEUE before running. Any new processes that arrive will not be added to the QUEUE. It is non-preemptive in nature because each process runs to completion. First, the algorithm allocates all burst times to data structure (BT) and all arrival times to data structure (AR). The first process that arrives runs to completion, then we calculate the average burst time of the remaining processes and use SJF to run all of the processes with burst times less than or equal to the average. It then counts the remaining processes, divides them by four, groups them, and divides them according to their groups, which are determined by the number of processes in the queue. As indicated in Figure 1, the CPU will be assigned to the processes by alternating between LJF and SJF.

The Pseudocode of the Proposed Modified SJF CPU Scheduling Algorithm

1. Let Q be a ready queue of available process PI, P2, P3… Pn.
2. Compute b_{min} = min burst time (Q), the minimum burst time for processes in Q.
3. Find process Pi ∈ Q having burst time b_{min} i.e. $b_{min} = (BT(P_i)_{i = 1 \text{ to } n})$
4. compute abt = $\frac{\sum_{k=2}^{n} BT(pk) - b\ min}{n-1}, Q_1 = Q - \{P_i\}$
5. let $Q_2 = \{p_j | p_j \in Q_1, BT(P_j) \leq abt\}$. Schedule processes Q_2 using SJF
6. Let $Q_3 = Q_1 - Q_2$, Sort Q_3 in decreasing order of burst time
7. Let Q_4 = processes Q_3 sorted in decreasing order of burst time
8. Schedule $Q_5 = \frac{1}{4}$ size of (Q_4)
9. Sort Q_5; in ascending order
10. Let x=abs $\left(\frac{size\ of\ (Q_5)}{4}\right)$
11. Let y= size of (Q_5)
12. Let j = 0
13. Let g= groupProcess(n)
14. For i 0 to g
    If i % 2 == 0
    Schedule Q_5[j] : Q_5 [j+x] using SJF
        j = j+x
    Else Schedule Q_5 [y-1]: Q_5 [(y-l)-x] using LJF
        y = (y-1) -x
    End if
    End for
15. Schedule the remaining processes using SJF.
16. Calculate AWT, ATAT of all processes
17. Stop

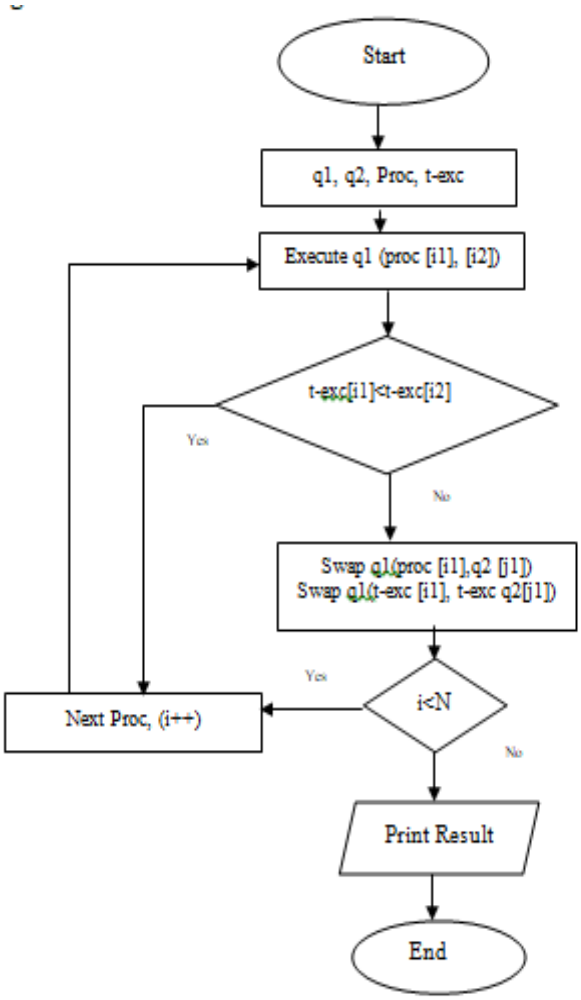

Figure 1: Flowchart of the Proposed Modified SJF CPU scheduling algorithm.

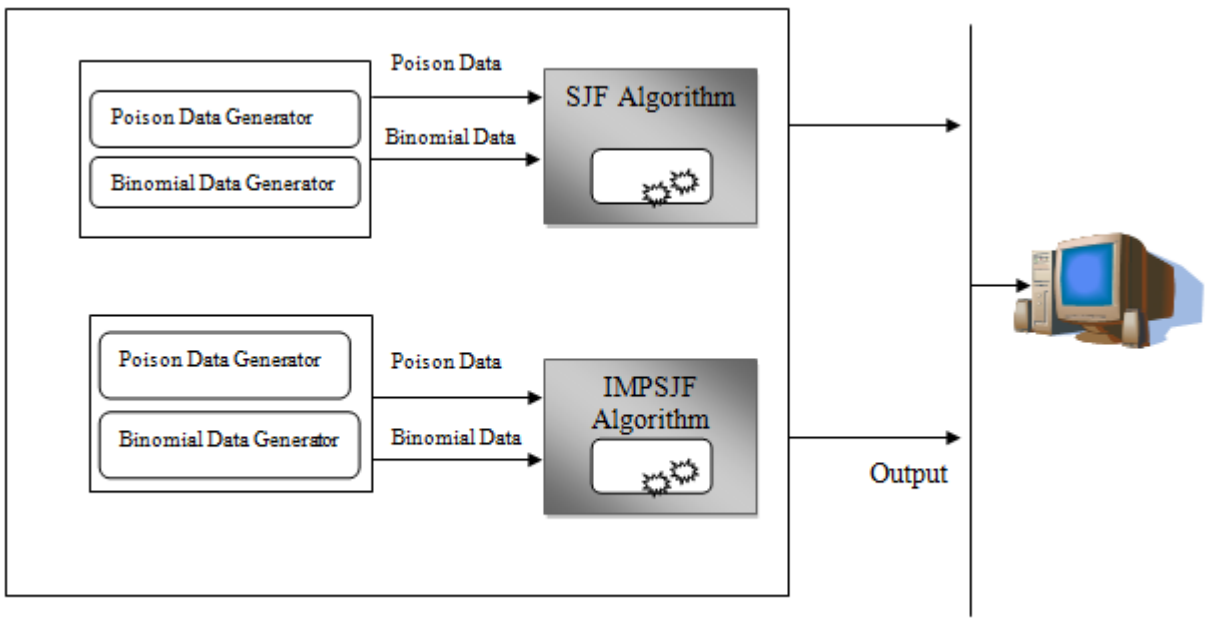

Figure 2: Overall System Architecture

The system takes in a number of processes N as input, the statistical distribution (Binomial and Poisson) used in generating burst times and the arrival time which uses it for both the SJF

and modified SJF CPU scheduling algorithms. The generated Processes will be allocated to the algorithms. The SJF will treat the processes according to the one that has the least burst time till it gets to the process with longest burst time while in the modified SJF, the least burst time will run to completion, it will then compute the average for the remaining burst time, after which the CPU will be assigned to processes that are less or equal to the average. Then remaining processes will be divided into four and grouped into processes for further division according to the number of processes input. The largest burst time will have the CPU to run using LJF and the next group will be run using SJF, it keeps alternating depending on the number of processes inputted at the beginning. The criteria will be evaluated and evaluated results will also be displayed on the computer screen. The equations below give the scheduling criteria for the evaluation of the result:

$$\text{Waiting Time} = \text{time first schedule} - \text{arrival time} \quad (1)$$

$$\text{Average Waiting Time (AWT)} = \frac{\text{sum of all processes waiting time}}{Number\ of\ Process} \quad (2)$$

$$\text{Turnaround Time (i)} = \text{burst time(i)} + \text{waiting} \quad (3)$$

$$\text{Average Turnaround Time (ATAT)} \ \frac{\text{sum of all processes Turnarround time}}{Number\ of\ Process} \quad (4)$$

$$\text{Starvation} = \text{waiting Time for bigger process} \quad (5)$$

RESULTS AND DISCUSSION

Results

Following all steps of the proposed algorithm, we will illustrate how it works with the aid of an example. Given 5 processes with their burst times and all processes arriving at time zero as in Table 1, the proposed algorithm works as follows

Table 1: Showing arrival and burst time

Process ID	Burst Time
P0	4
P1	13
P2	9
P3	5
P4	7

P0 has the least burst time, it will be moved to the READY queue and the CPU will be allocated to it. After it finishes the execution of P0; the remaining processes PI, P2, P3 and P4 will be sorted according to their burst time; PI (13), P2 (9), P3 (5) and P4 (7). The average for the burst time will be taken (13+9+5+7) / 4 = 34/4 which is 8.5. Since P3 (6) and P4 (7) are less or equal to the average that means P3 and P4 will be assigned the CPU because their burst time is not greater than 8.5. The remaining processes will be divided into two; given the CPU to the one with the highest burst time, the next process will be P1 and Finally, P2 as in Figure 3.

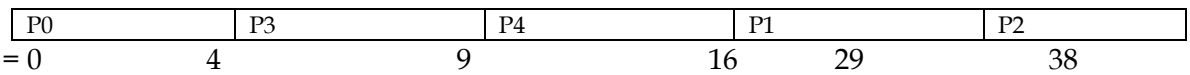

Figure 3: IMPSJF Gant Chart

Table 2: Waiting Time of Processes using IMPSJF

Process ID	Burst Time	Waiting Time
P0	4	0
P1	13	16
P2	9	29
P3	5	4
P4	7	9

IMSJF = (0+16+29+4+9) / 5 = 11.6

For SJF using table 4.1 processes P0 will be given to the system first because it has the least burst time followed by P3, P4, P2, P1 as shown in Figure 4.

P0	P3	P4	P2	P1

t = 0 4 9 16 25 38

Figure 4: SJF Gantt Chart

Table 3: Waiting Time of Processes using SJF

Process ID	Burst Time	Waiting Time
P0	4	0
P1	13	25
P2	9	16
P3	5	4
P4	7	9

SJF = (0+25+16+4+9) / 5 = 10.8

Table 3 shows the average waiting time of SJF (10.8) is better than that of IMSJF (11.6) but IMSJF reduces the waiting time for processes P1 which has the largest burst time. P1 SJF waiting time is = 25 and P1 IMSJF waiting Time = 16.

Table 4: showing the WT and TAT

Process	Arrival Time	Burst time	IMPSJF WT	IMPSJF TAT	PROCESS	BURST TIME	SJF WT	SJF TAT
P0	1	3	0	3	P0	3	0	3
P1	4	6	3	9	P1	6	3	9
P2	6	14	9	23	P2	14	9	23
P3	5	21	23	44	P3	21	40	61
P4	6	17	44	61	P4	17	23	40

Table 4 depicted some processes P0, P1, P2, P3, P4 arrival time, burst time, waiting time and turnaround time for SJF and IMPSJF while Table 5 shows the average waiting time (AWT) and average turnaround time (ATAT).

Table 5: showing AWT and ATAT

Algorithms		Average Waiting Time	Average Turnaround Time
SJF		15.0	27.20
IMPSJF	15.8	28.0	

Figure 5 shows comparison of SJF and IMPSJF in a chart.

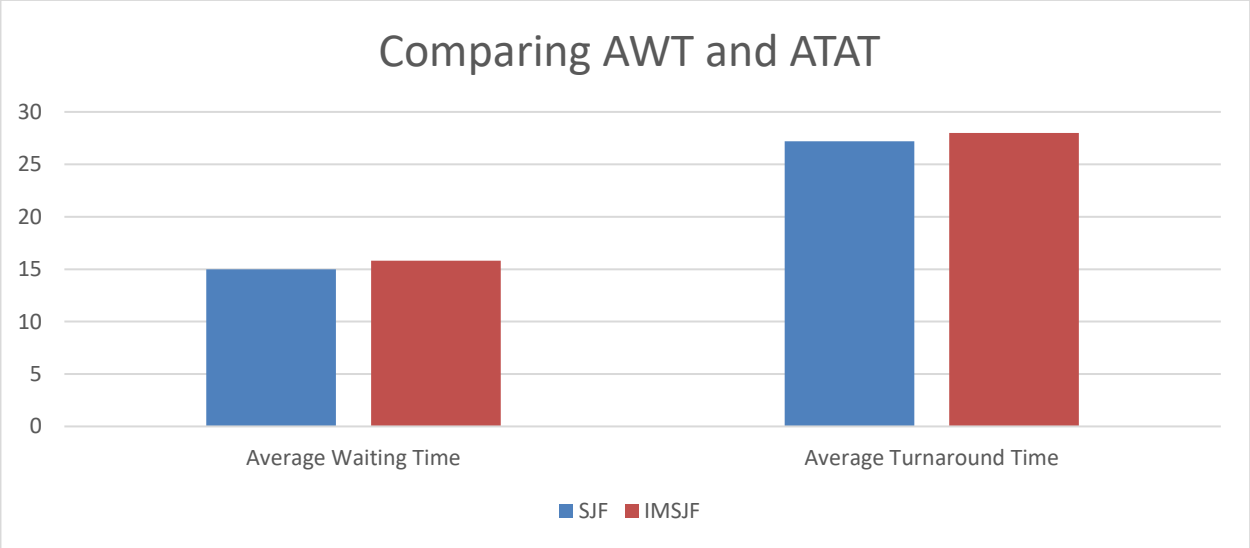

Figure 5: Graph Comparing AWT and ATAT

Preliminary of the Simulation Program

All of the experiments were carried out in a single processor simulation environment. The implementation was completed primarily with the Java programming language and NetBeans. Four (4) Java classes were used to implement the complete application. The first class was called CPU. The second class, GenericStimulator.java, was used to show and collect the simulation's data. The third is a java class called StochasticVariableGenerator.java, and it is in this class that both Binomial and Poisson distribution data generation are handled. Finally, the fourth class, ImprovedSJF.java, is where the new algorithm is implemented.

The implementation of the stimulation program uses the graphical user interface as a means of data communication exchange with the user. Basically, the user is expected to supply the number of processes to be simulated at the first input to the program when it is lunched. This input will be sent to the part of the program where the specified distributions (Binomial and Poisson) are used in generating both the arrival and burst times. After this data have been generated, the user will again be prompted to click on a start button to initiate the process. These multiple running of the stimulation is making closer comparison over their results and then draw an appropriate conclusion on the result. Figure 6 captures a snapshot of how the simulation was launched using the NetBeans IDE. The interface being displayed can be regarded as the regular command prompt interface. The result of the simulation is available both at the command prompt interface as well as an initially specified data file.

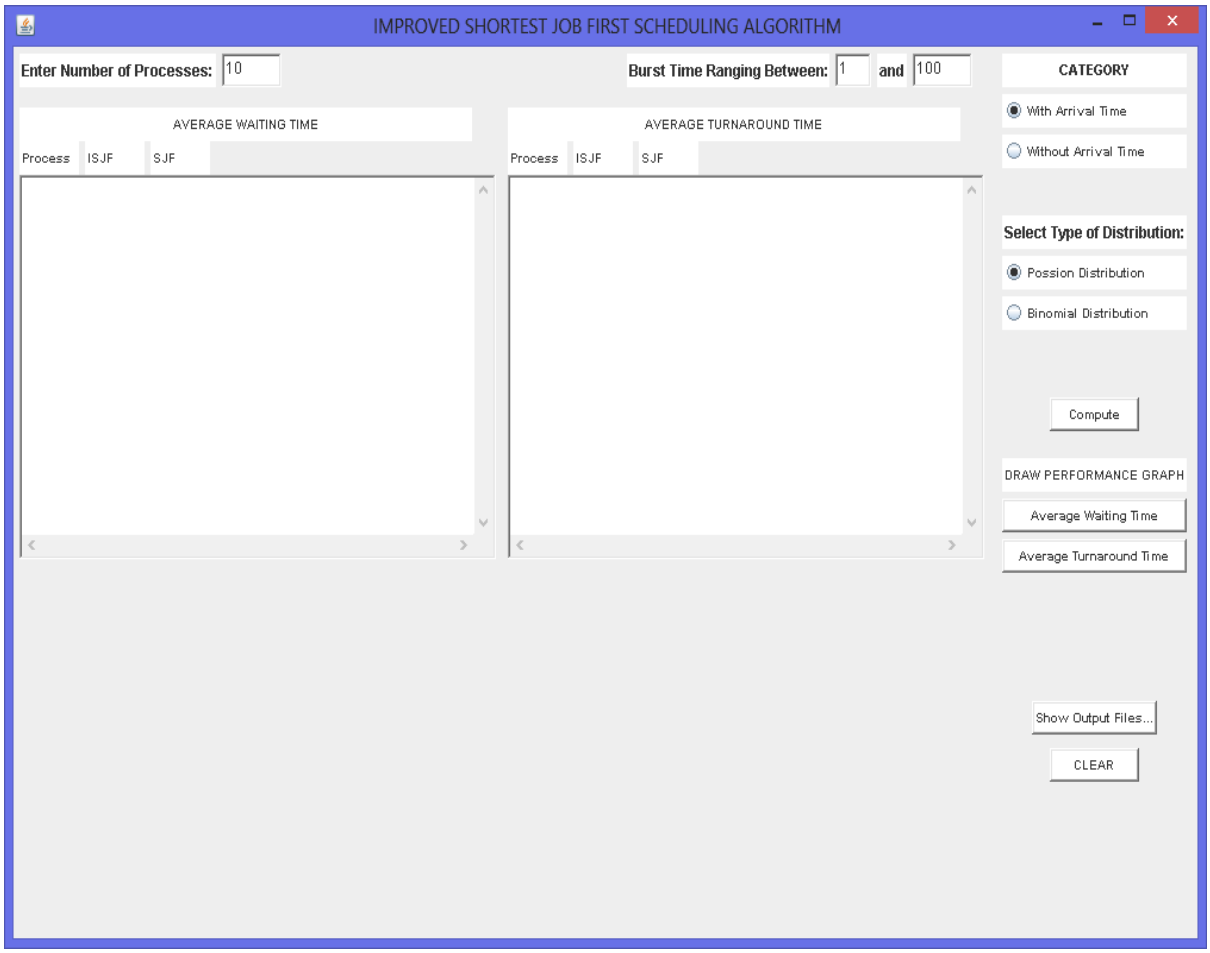

Figure 6: Stimulation Program request for User Input

The Figure 6 depicted the illustrative user interface initiated by the program at first stage. The interface prompts users to input the number of process to be executed and the type of distribution which is either Poisson or Binomial as the study specified. After the user supplied the needed input, then users can click on proceed button to generated the appropriate processes required for the next execution.

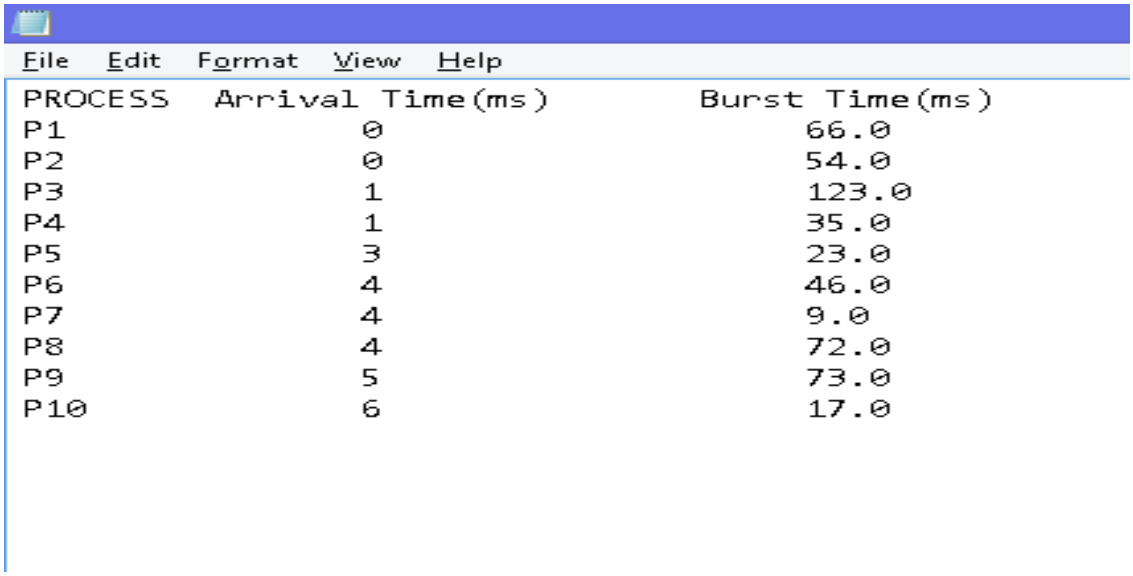

Figure 7: Generated Burst Time and Arrival Time Using Poisson distribution

Figure 7 displayed the generated arrival time and burst time using Poisson distribution of 10 inserted processes and a specified given range of burst time.

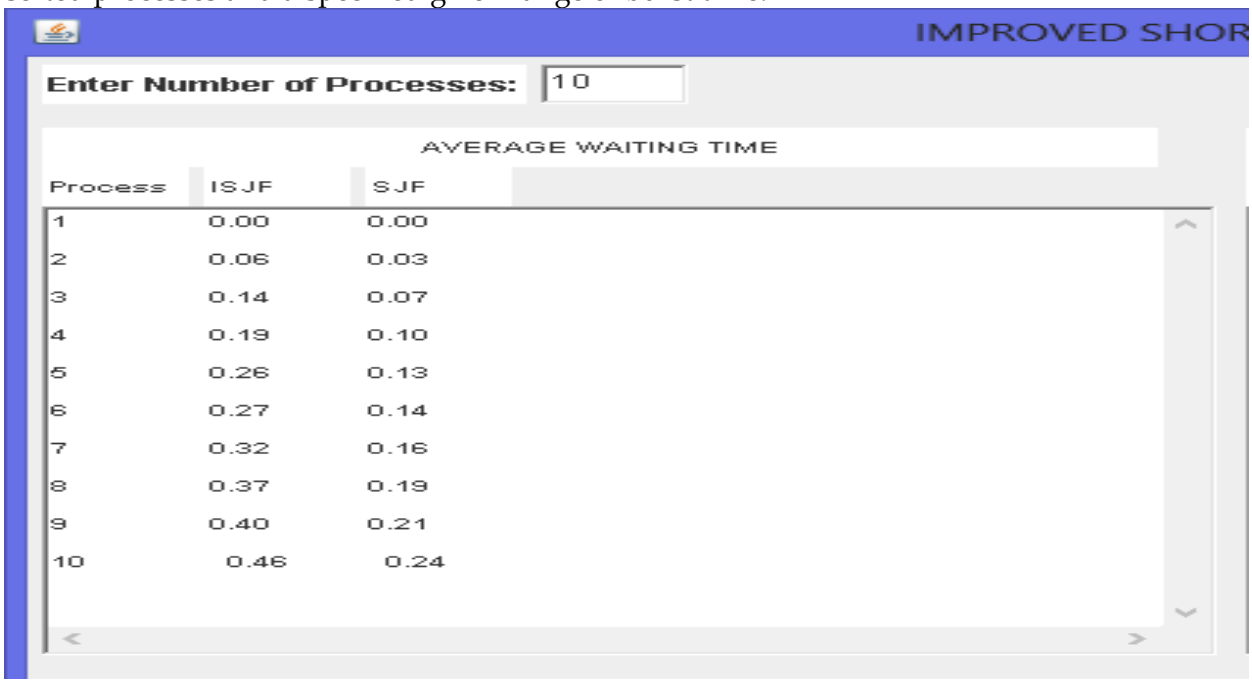

Figure 8: Generated Waiting Time of ISJF and SJF using Poisson distribution

Figure 8 shows the number of processes simulated by the program using Poisson distribution, each process has the figure of a given waiting time as generated during the execution. Based on the input made by the user as shown in Figure 8, the simulation gets executed and then displays the result. In this instance of the simulation, 10 processes were entered by the user and their corresponding data were generated.

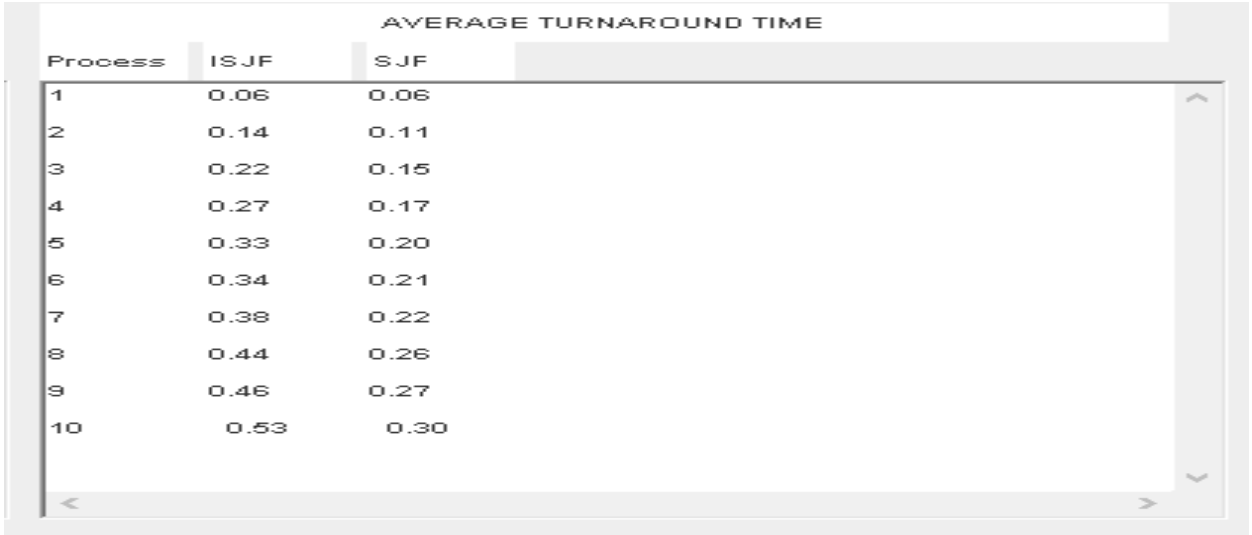

Figure 9: Generated Turnaround Times of ISJF and SJF Using Poisson distribution

Figure 9 shows the result of turnaround time generated by each process using Poisson distribution, each process has the figure of a given turnaround time as generated during the execution. In this instance of the simulation above, 10 processes were entered by the user and their corresponding result was generated.

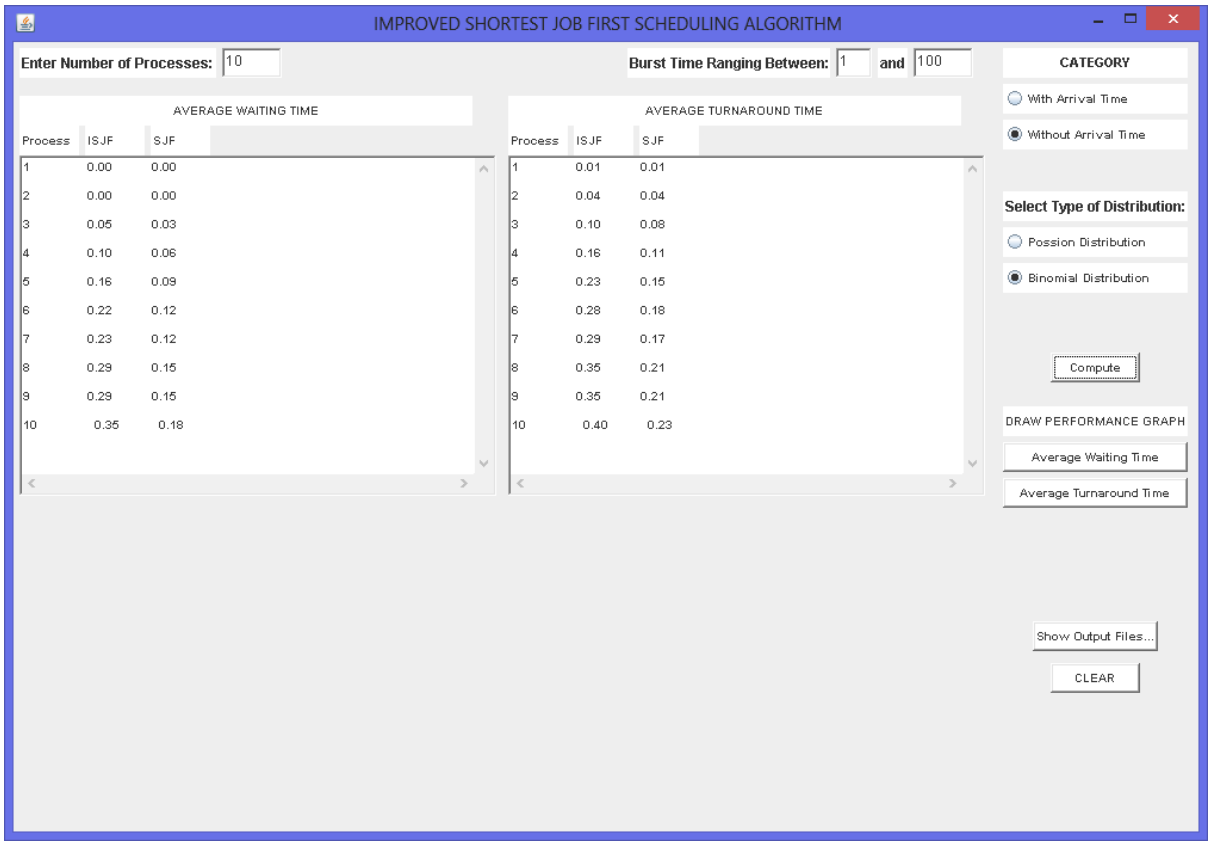

Figure 10: Computed Waiting Time and Turnaround Time of SJF AND IMPSJF

Figure 10 displayed the computed figures of the waiting time and turnaround time of a given number of processes using Binomial distribution function.

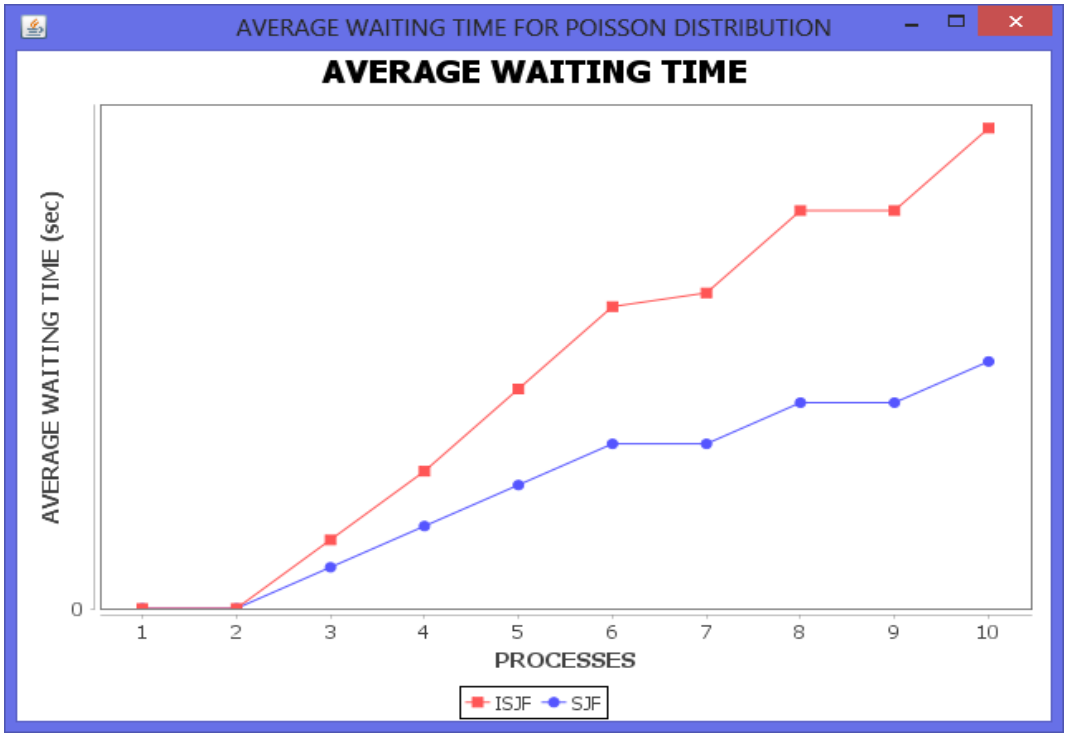

Figure 11: Average Waiting Using Poisson distributions

Figure 11 illustrates the graphical simulated result from the program. The result showed the average waiting time of 10 given processes using Poisson distribution functions.

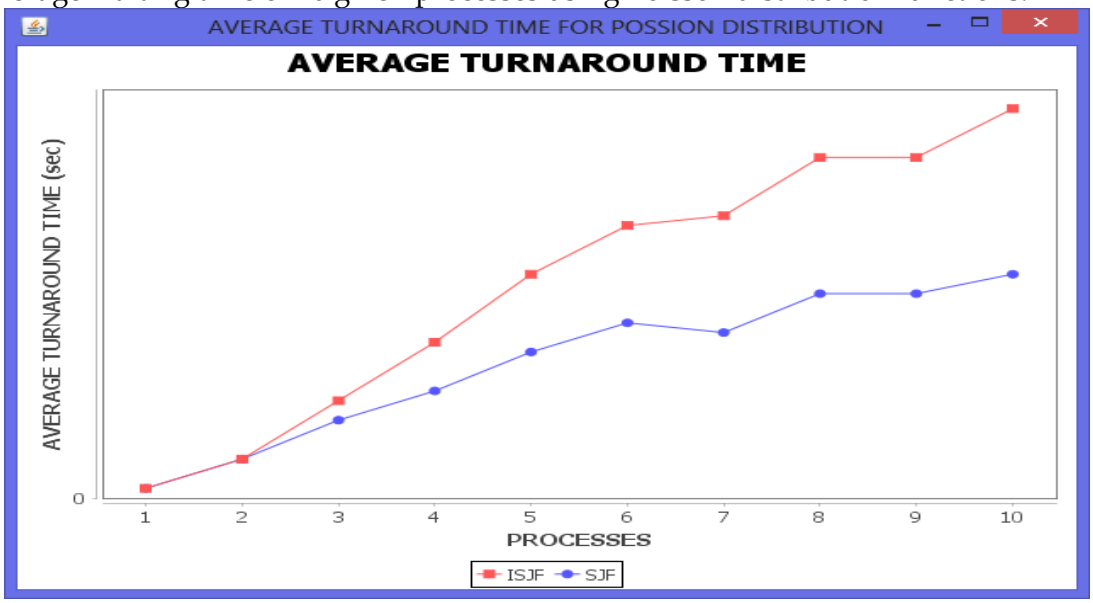

Figure 12: Average Turnaround Time Using Poisson distributions

Figure 12 illustrates the graphical simulated result from the program. The result showed the average turnaround time of 10 given processes using Poisson distribution functions.

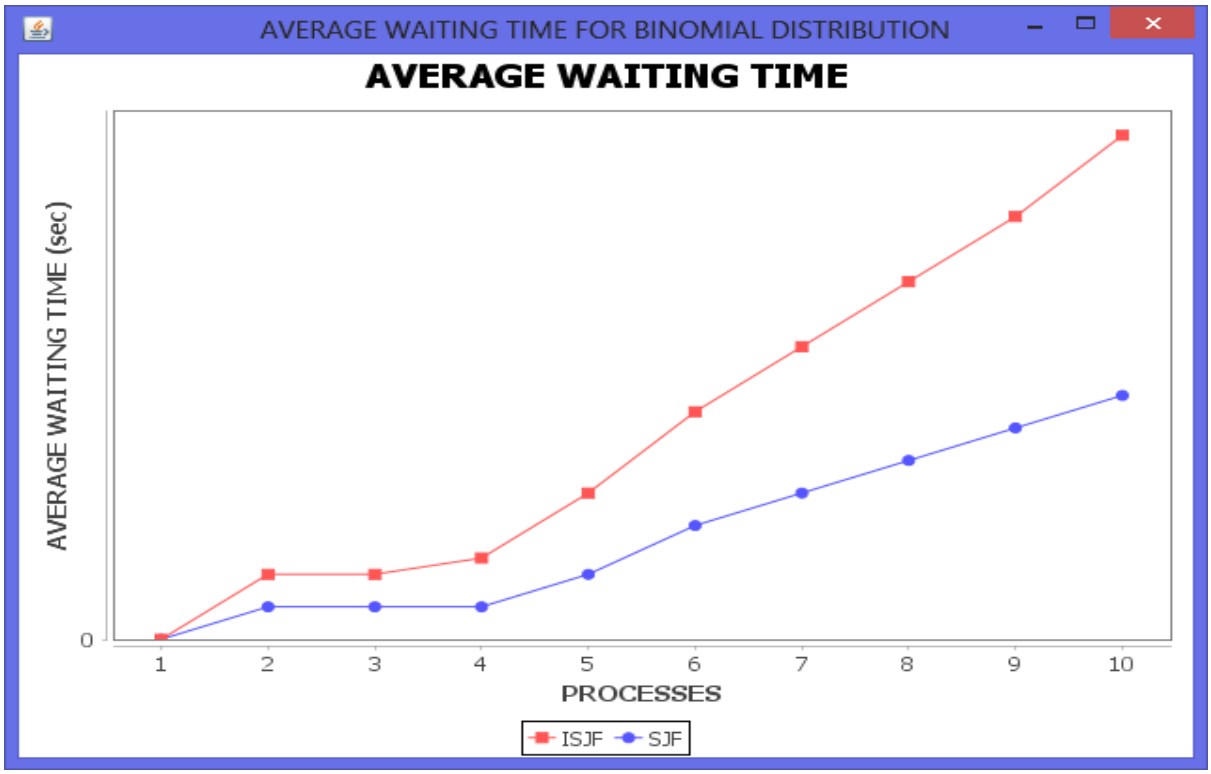

Figure 13 Average Waiting Time Using Binomial distributions

Figure 13 illustrates the graphical simulated result from the program. The result showed the average waiting time of 10 given processes using Poisson distribution functions.

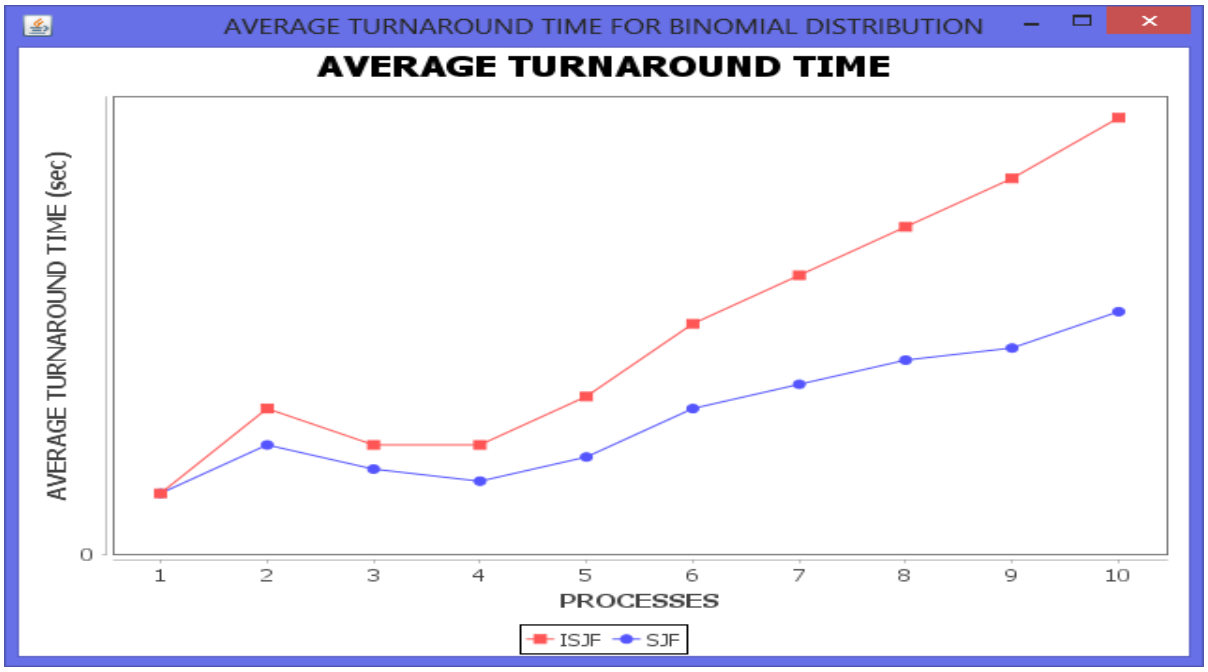

Figure 14 Turnaround Time Using Binomial distributions

Figure 14 shows the average turnaround time of 10 given processes using Poisson distribution functions.

Discussion

The abovementioned experiment shows that the SJF's overall waiting time, average turnaround time, and context switches are comparable to the suggested approach. Longer processes are also less likely to be starved because they wait for a set amount of time before being allocated the CPU, as opposed to the SJF. This represents the highest CPU utilization and the shortest response time. In terms of not starving processes, we can say that the suggested approach is far more efficient than the conventional technique.

Comparison with other literatures

The developed algorithm IMSJF was compared with the works of Shafi et al. (2019) (Amended Dynamic Round Robin (ADRR)), Elmougy et al. (2017) (Shortest job first and Round robin with Dynamic Quantum hybrid algorithm (SRDQ) and Ali et al. (2020) (HYbrid Round Robin scheduling mechanism(HYRR). These reviewed algorithms were compared with the proposed IMSJF, in terms of AWT, ATT and number of CS using the initially generated processes with Poisson distribution in Figure 7. Figure 15 and 16 showed the result of the comparison for AWT and ATT respectively.

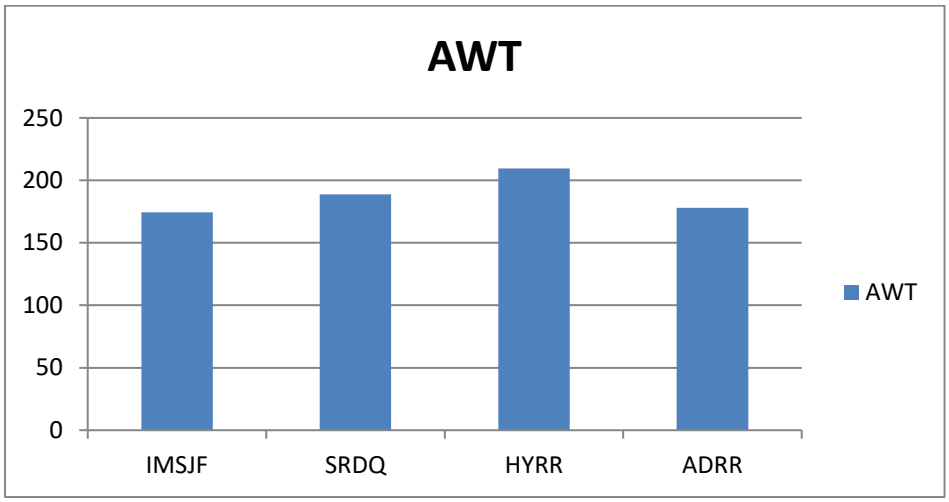

Figure 15: Comparison of AWT for each Algorithm

Figure 15 clearly showed that IMSJF has a smaller AWT with 174.4, ADRR, SRDQ and HYRR have 178, 188.8 and 209.5 respectively. In terms of ATT, IMSJF still lead the pack with the least ATT of 226.8 whereas the rest of the algorithms which include ADRR, SRDQ and HYRR are having 230, 240.6 and 261.3 respectively as shown in Figure 16.

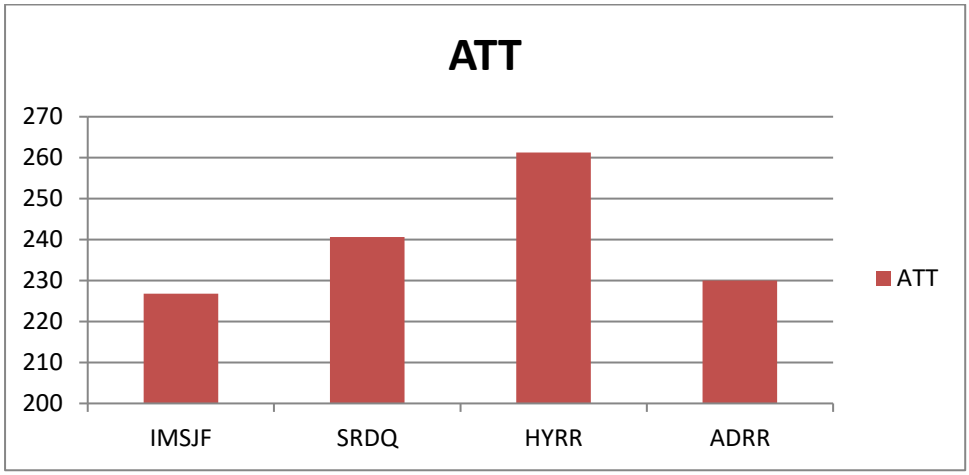

Figure 16: Comparison of ATT for each Algorithm

Table 6 compares the three (3) parameters used for evaluating the algorithms. These parameters are AWT, ATT and number of CS. The result showed that IMSJF performed almost optimally with 174.4 AWT, 226.8 ATT and 0 CS.

Table 6: Comparisons of the Algorithms using the three Evaluated Metrics

	IMSJF	SRDQ	HYRR	ADRR
AWT	174.4	188.8	209.5	178
ATT	226.8	240.6	261.3	230
No. of CS	0	1	5	0

CONCLUSION

Different CPU scheduling algorithms abound, however, this study was limited to alleviating the starvation problem of SJF algorithms. Specifically, SJF and the suggested IMPSJF scheduling algorithms were compared; additionally, Binomial and Poisson distributions were utilized as statistical data generation distributions in the research; and it is obvious from the results that in IMPSJF:

a. Larger processes are prevented from starvation
b. Waiting time for larger processes are minimized

Because IMPSJF is non-preemptive, no context transition happens, and there is no need to save the present states of the processes, which is considered as an overhead in preemptive scheduling policies. Finally, data analysis and simulation were performed, and the IMPSJF performed better in decreasing starvation concerns of large burst time processes when they arrived in the ready queue. The results also reveal that SJF outperforms IMPSJF marginally. When it comes to starving, both SJF and IMPSJF might have the same result at times, and in some circumstances, IMPSJF is somewhat better than SJF. After comparing the algorithms, it is determined that IMPSJF is superior in terms of minimizing the degree of starvation. We suggest that future study in this area focus on improving the average waiting time by comparing IMPSJF with more parameters and making the method recursive.

REFERENCES

Ali K. F., Marikal A., and Kumar K. A. (2020). A Hybrid Round Robin Scheduling Mechanism for Process Management. International Journal of Computer Applications, 177 (36): 14-19.

Elmougy, A., Ramakrishna, M., & Pattabhi, G. R. (2017). Dynamic quantum hybrid algorithm (SRDQ) for Operating Systems. *International Journal of Innovative Technology and Research , 1* (1), 103-109.

Saroj, H. & Roy, K. C. (2013). Adaptive Round Robin scheduling using shortest burst approach, based on smart time slice. *International Journal of Data Engineering 2013(11*), 175-181.

Shafi, U., Shah, M., Wahid, A., Kamran, M., Javaid, Q., Asghar, M., & Haider, M. (2019). A Novel Amended Dynamic Round Robin Scheduling Algorithm for Timeshared Systems. *International Arab Journal of Information Technology*, *16*(4), 1–9.

Silberschatz, A., Galvin, P. B. & Gagne, G. (2018). *Operating System Concepts.* (10th, Ed.) John Wiley and Sons Inc., New Jersey, USA.

Tanenbaum, A. S. and Bos, H. (2015). Modern Operating Systems (4th Ed.), Upper Saddle River, New Jersey, USA: Pearson Education, Inc.

Teraiya J. R. and Shah A. (2018). Comparative Study of LST and SJF Scheduling Algorithm in Soft Real-Time System with its Implementation and Analysis. International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, India, 19-22 September 2018.