A k-Factor CPU Scheduling Algorithm

Sumedha Garg

Department of Computer Science
Student of Mathematics and Computing
Banasthali Vidyapith, Rajasthan, India
sumedhagarg@hotmail.com

Deepak Kumar

Department of Computer Science
Faculty of Mathematics and Computing
Banasthali Vidyapith, Rajasthan, India
deepakkumar@bansthali.in

Abstract— **In a multi-tasking environment, it is crucial to optimize the CPU scheduling process. Several algorithms have been proposed and are in use to schedule processes on CPU. This paper aims to evaluate shortest job first (SJF) algorithm and propose an algorithm to overcome the shortcomings of SJF. The simulation results yield useful data for comparison and evaluation. The performance metrics include average turnaround time and average waiting time. The numbers of processes have been varied in the range 10 to 100000, which provides actual scenarios inside a system. For evaluation, simulations have been run many times, between 10 to 100 runs. The results indicate that the starvation created by SJF for longer processes is not present in the proposed algorithm, making it a suitable choice for implementation.**

Keywords—process scheduling, shortest job first, SJF, K Factor, starvation, waiting time, turnaround time

I. Introduction

Over the last few decades, the role of Operating systems has become more crucial involving complex tasks for better CPU utilization. To efficiently manage the processes, scheduling algorithms are required. Many such algorithms [1] have been proposed like first come first serve (FCFS), round robin (RR), shortest remaining time next (SRTN), shortest job first (SJF), etc. The primary objectives of scheduling algorithms are:

i. Decrease average starvation for processes.

ii. Assure fairness

iii. Processing the priority process before others and

iv. Reduce the chance of infinite postponement.

The available algorithms try to work on these limitations. SJF algorithm is used extensively and provides excellent results but suffer from the problem of starvation. In this paper, we have tried to overcome this issue by proposing a novel algorithm.

The primary source of study for this work is the book Operating System Concepts by Silberschatz, Galvin, and Gagne [2]. They have stated that scheduling is a vital functionality of an operating system. It allows concurrent execution of processes, thereby maximizing the CPU utilization. The processes are chosen for execution, as per a scheduling algorithm.

State of the art is presented in the next section followed by a brief description of scheduling parameters and SJF algorithm in next sections. Section V presents the proposed algorithm. Results are discussed in next section followed by the conclusion. In the end, references to this work are provided.

II. State of the Art

In order to make the system in hand more efficient, many scheduling algorithms have been suggested over the time but these algorithms need to conform to some factors like finish process on time, minimize starvation, turnaround time and waiting time of the processes for resource allocation. By taking these factors under consideration the algorithms namely FCFS, SJF, Round robin and many others have been developed. But, as the number of requirements and the processes are increasing daily, researchers are putting their efforts to develop new algorithms. Some of the new researches in this field which accounts the growing need of the industry are mentioned below [11].

In Basit et al [3], shorter processes are assigned to the CPU as in SJF, but after some time, bigger processes can get the CPU in Enhanced SJF. The time interval is selected dynamically and is considered as an overhead. While in Muhammad et al [4], the approach behind optimized SJF is that on inception it chooses the job having the shortest burst time similar to SJF and executes it until a new job arrives in the ready queue, after which it decides whether or not to preempt the running process which matches the concept of SRTF. If half of the remaining burst time of the executing process is less than the burst time of the new process in ready queue, then executing process will not preempt but if it is not, then we will preempt the executing process and assign the CPU to newly arrived process. It also has an overhead of context switching and preemption.

Jain et al. [5] showed an Improved Round Robin algorithm with characteristics of Shortest Job First scheduling with changing time quantum. Similarly Yadav et al. [6] also proposed an algorithm which is a combination of SJF and RR algorithm. This algorithm worked better than pure RR.
Some of the most useful contributions in the field of operating systems and scheduling algorithms are made by Dhamdhere [10], Silberschatz and Galvin [2], Tanenbaum and Woodhull [9] and Stalling [8].

So based on these and many other researches and there limitations we decided to give bigger processes a chance at CPU before smaller processes by comparing twice of the life cycle of a small process by the life cycle of adjacent bigger process and work done by CPU.

III. Scheduling Parameters

Scheduling parameters are used to compare scheduling algorithms in multiprogramming systems. The criterion for the choice of suitable scheduling algorithm depends upon the following parameters:

A. CPU utilization [2]: It is the average percentage of the time, during which the CPU is busy. CPU utilization refers to a computer's handling of its resources or the amount of work done by the CPU.

B. Waiting Time [2]: It is the amount of time a process spends waiting in the ready queue to receive CPU resources. Mathematically waiting time is expressed as turnaround time, minus the burst time.

C. Turnaround time [2]: It is the number of processes completed in a given unit of time or the average number of successful processes completed over a specified period of time.

D. Fairness [2]: It refers to the degree of fair consideration given to process, as all the processes must get fair and equal chance to execute

E. Starvation [2]: Starvation occurs when a process is denied necessary resources again and again without which the process can never finish its task

IV. Shortest job first (SJF) algorithm

Shortest Job First (SJF) [1] is a scheduling algorithm that selects the process from ready queue with the smallest burst time to execute. When the executing process completes its execution it is terminated and gets removed from the ready queue. Then a new process having the smallest burst time is selected for execution from the ready queue. If any two processes in the ready queue having the same burst time are detected, then FCFS is used to break up the tie [4]. Advantages of the shortest job first over other scheduling algorithms are listed below.

A. The simplicity of the algorithm.
B. Its average waiting time is less than other scheduling algorithms [7].
C. Its throughput is quite better than the other scheduling algorithms.
D. High CPU utilization
E. It increases process throughput regarding the number of processes run to completion in a given amount of time.
F. SJF chooses the job with the smallest burst time making sure the CPU's availability for other processes in the queue as soon as the current process ends. This hinders the shorter processes from suffering behind bigger processes in the ready queue for a large amount of time.

Besides its numerous advantages, it has many disadvantages as well, they are:

A. Starvation is a problem in SJF where a process is denied necessary resources which are required to finish its task.
B. Infinite postponement occurs when processes which will need a long time to complete its task does not get CPU as short processes are added continuously.
C. Not all the processes have an equal chance of being executed as the other because this algorithm is polarized towards the smaller processes which introduce unfairness in the system.

V. proposed K factor algorithm

The primary goal of the proposed algorithm is to remove or minimize the limitations posed by the SJF algorithm. The limitations which we have worked on are

1. Starvation of longer processes
2. There is polarization towards shorter processes, i.e., unfair to longer processes.

The algorithm is as stated below:

Step 1: Initialization

```
a.  Variable K=2
b.  For i = 0 to n
      i.   P[i] = Random Number Generator(1,100),
           Where p[i] is the burst time of process i in
           ready queue,
      ii.  A[i]=0, where A[i] is the arrival time of
           process i
```

Step 2: Align the processes in the ascending order of their burst time in the ready queue.

Step 3: Allocate the CPU to the first process in the ready queue, P[1] for the time duration of its burst time.

Step 4: Remove the process from the ready queue which has been allocated the CPU.

Step 5: Select the first two processes in the ready queue, P [1] and P [2] (where P [1] < P [2]). Now let us assume that CPU has worked for some time say t, represented in the Gantt chart then

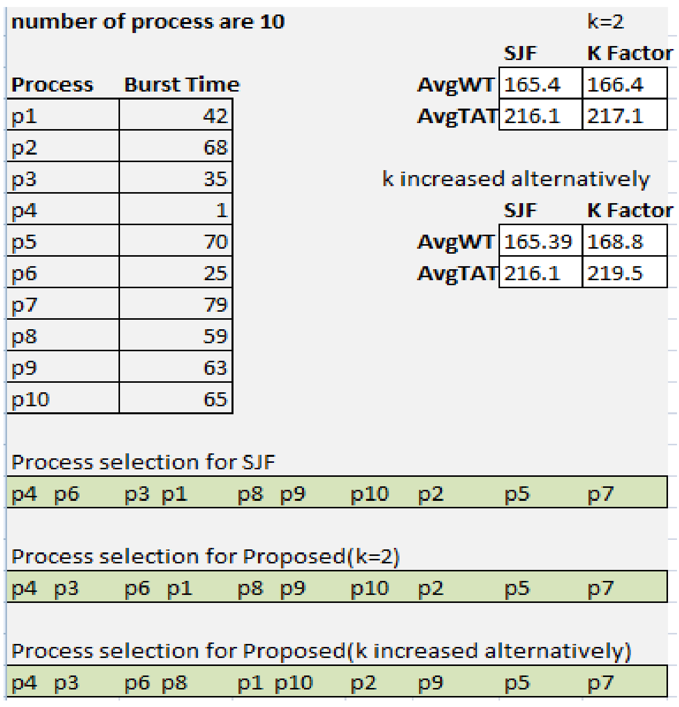
```
a.  If, (K* P [1]) > (t + P [2])
            Process P [2] gets the CPU
    else
            Process P [1] gets the CPU

b.  Increase the value of K alternatively.
```

Step 6: Go to step 4 if there are processes in ready queue else exit.

In the above algorithm, we have used a factor called K whose value is initialized with 2 and incremented alternatively in a loop.

Now we don't necessarily need to increment the value of K alternatively. We can either forgo this step or merely increase the value of K with every cycle by one, but both the choices would provide different process selection.
If we increase the value of K continuously, then the algorithm will deviate the value of average turnaround time and waiting time from SJF. Hence we do not prefer this option but if we increase the value of K alternatively then it will only deviate the values for a small number of processes, and therefore we explore this possibility.

Figure 1 shows sample run of SJF and K factor algorithm for ten processes with a value of K=2 for all the processes and value of K being incremented alternatively.



number of process are 10 k=2

Process	Burst Time
p1	42
p2	68
p3	35
p4	1
p5	70
p6	25
p7	79
p8	59
p9	63
p10	65

	SJF	K Factor
AvgWT	165.4	166.4
AvgTAT	216.1	217.1

k increased alternatively

	SJF	K Factor
AvgWT	165.39	168.8
AvgTAT	216.1	219.5

Process selection for SJF

p4 p6 p3 p1 p8 p9 p10 p2 p5 p7

Process selection for Proposed(k=2)

p4 p3 p6 p1 p8 p9 p10 p2 p5 p7

Process selection for Proposed(k increased alternatively)

p4 p3 p6 p8 p1 p10 p2 p9 p5 p7

Fig.1. Sample run for different K Factors

Table I showed the average of waiting time and turnaround time of SJF and proposed K factor algorithm against increasing number of processes. Here the two different algorithms were used to simulate k Factor algorithm, one with excluding step 5(b) of the algorithm and other including it.

TABLE I. WAITING TIME AND TURNAROUND TIME

Processes	Avg WT			Avg TAT		
	SJF	K Factor		SJF	K Factor	
		K=2	K incr altr		K=2	K incr altr
10	165.4	166.4	168.8	216.1	217.1	219.5
50	789.2	789.22	790.94	838.56	838.58	840.3
100	1584.62	1584.62	1585.28	1632.1	1632.1	1632.76
500	8314.568	8314.568	8314.856	8364.764	8364.764	8365.052
1000	16309.35	16309.35	16309.57	16358.6	16358.6	16358.82
5000	86477.9	86477.91	86477.91	86529.16	86529.18	86529.18
10000	171133.8	171133.9	171133.9	171184.7	171184.8	171184.8
25000	426002.6	426002.8	426002.8	426053.4	426053.6	426053.6
50000	848878.8	848878.8	848878.8	848929.4	848929.4	848929.4
75000	1271030	1271030	1271030	1271080	1271081	1271081
100000	1692034	1692036	1692036	1692085	1692086	1692086

From table I and figure 1 we can infer that for a small number of processes we can skip step 5(b) of the given algorithm but the selection of processes from the ready queue will vary less between SJF and K Factor. But we choose to include step 5(b) for a more significant number of processes as there is no significant difference between average waiting times and turn round times, but process selection varies more prominently.

Hence the analysis can be summed up as:

TABLE II. PROCESS SELECTION FOR K FACTORS

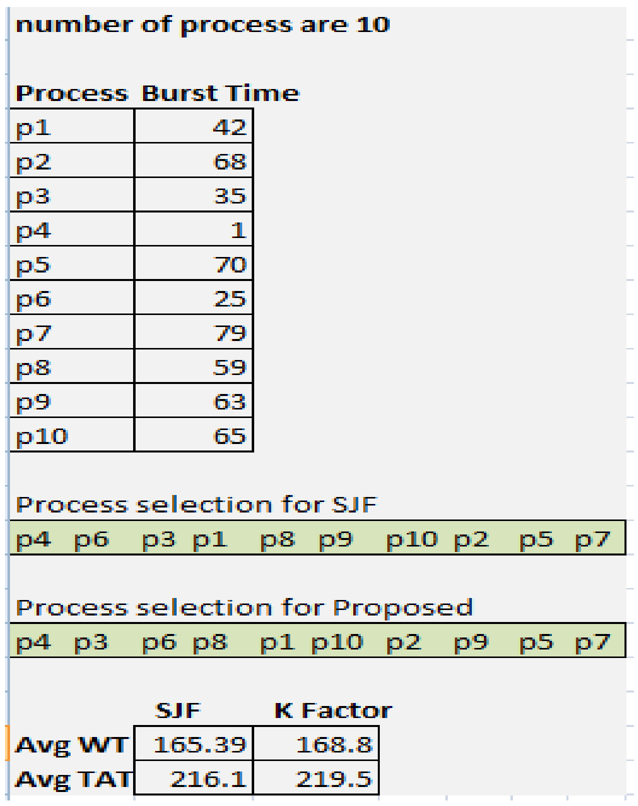
No. of Processes	K Factor used	Process Selection Variation	Approximation to SJF
5-500	2	Less	Faster
>500	Increases alternatively	More	Slower

Therefore, after analyzing the above table, we choose to include step 5(b) in the algorithm.

Example of proposed K factor algorithm

This example illustrates how our proposed algorithm, K Factor works and compares it with SJF regarding process selection from ready queue. The burst time for processes is generated randomly from 1 to 100. We have taken an example of 10 processes in the ready queue.

In this example, we demonstrate how both the algorithm choose processes from ready queue to allocate CPU after sorting all the process in ascending order of their burst time.



number of process are 10

Process	Burst Time
p1	42
p2	68
p3	35
p4	1
p5	70
p6	25
p7	79
p8	59
p9	63
p10	65

Process selection for SJF

p4	p6	p3	p1	p8	p9	p10	p2	p5	p7

Process selection for Proposed

p4	p3	p6	p8	p1	p10	p2	p9	p5	p7

	SJF	K Factor
Avg WT	165.39	168.8
Avg TAT	216.1	219.5

Fig.2. Sample run of SJF and K Factor algorithm

VI. SIMULATION SETUP AND RESULTS

To simulate the above algorithm, we have used a pseudo-random number generator to generate the burst time of a process. We have also divided the processes into three categories based on their burst time.

TABLE III. PROCESS CATEGORIZATION

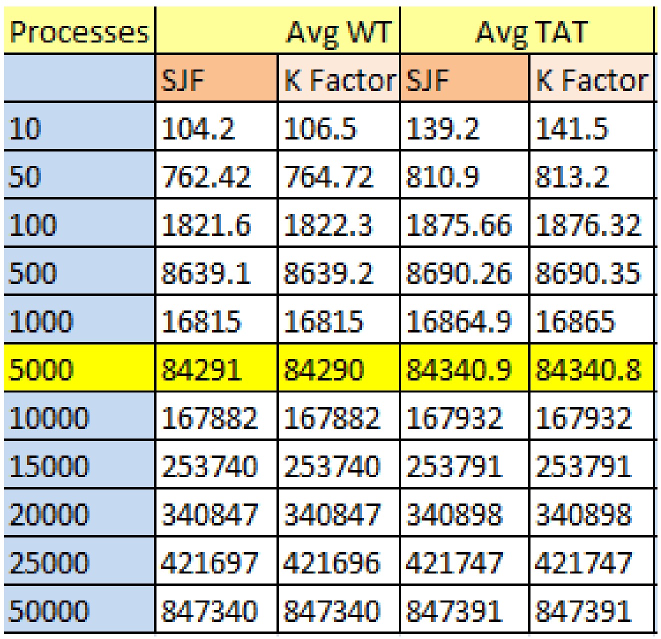
Burst Time	Category
1-15	Short Processes
16-50	Medium Processes
51-100	Large Processes

We have chosen c language to simulate the algorithm as it fulfilled all the requirements and due to the familiarity of this language. We simulated the output directly into excel sheets for the ease of understanding.

By giving the above algorithm, we do not aim to outsmart SJF but to provide an alternate algorithm of SJF with fewer limitations. The below table's show how the algorithm approximates and in some cases gives better results than SJF as we increase the number of processes in the ready queue.

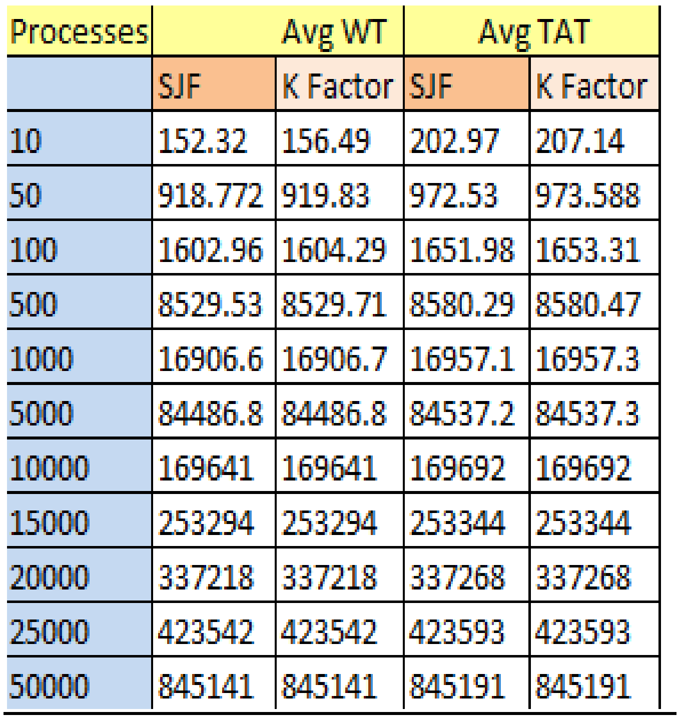
In the following tables, the average waiting time of SJF algorithm is compared with the average waiting time of the proposed algorithm, K Factor against a different number of processes. Similarly, Average turnaround time is also compared in the following tables. Table IV, V, VI and VII shows the output averaged over one, ten, fifty and hundred simulation runs.

TABLE IV. AVERAGED OUTPUT



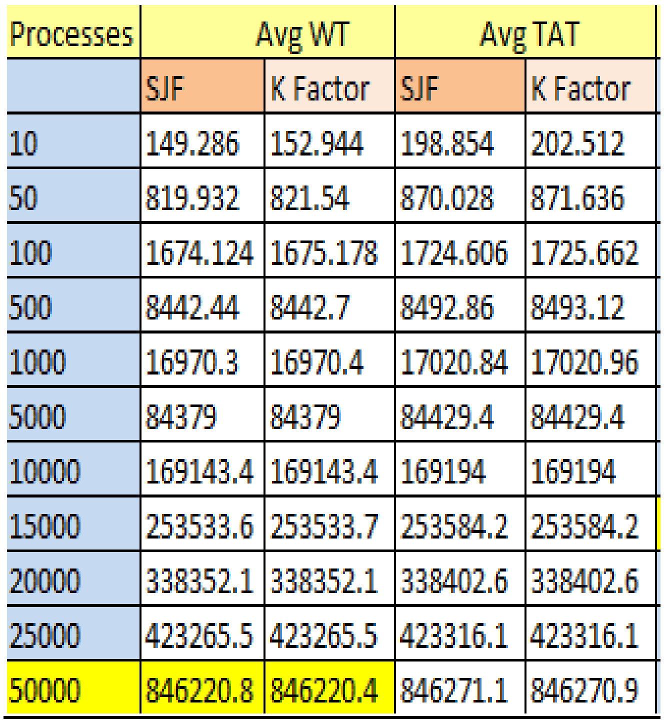
Processes	Avg WT		Avg TAT	
	SJF	K Factor	SJF	K Factor
10	104.2	106.5	139.2	141.5
50	762.42	764.72	810.9	813.2
100	1821.6	1822.3	1875.66	1876.32
500	8639.1	8639.2	8690.26	8690.35
1000	16815	16815	16864.9	16865
5000	84291	84290	84340.9	84340.8
10000	167882	167882	167932	167932
15000	253740	253740	253791	253791
20000	340847	340847	340898	340898
25000	421697	421696	421747	421747
50000	847340	847340	847391	847391

TABLE V. AVERAGED OUTPUT OVER TEN RUNS



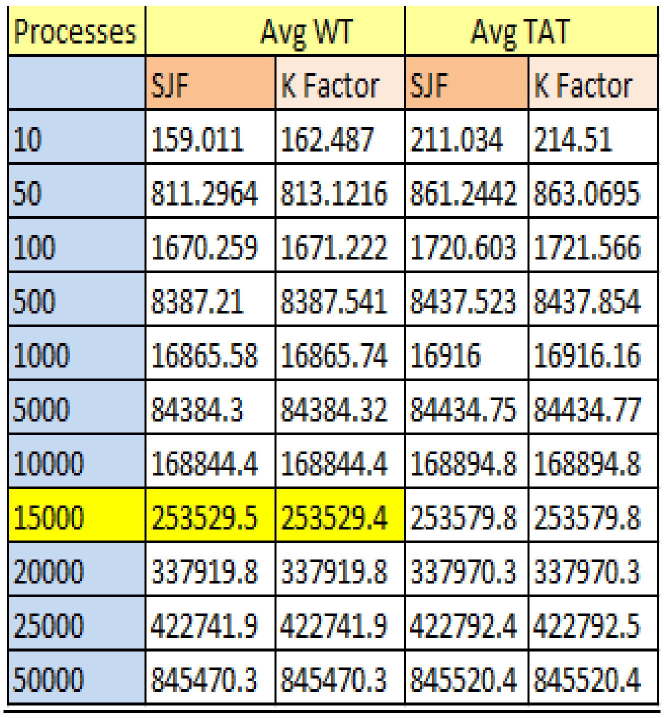
Processes	Avg WT		Avg TAT	
	SJF	K Factor	SJF	K Factor
10	152.32	156.49	202.97	207.14
50	918.772	919.83	972.53	973.588
100	1602.96	1604.29	1651.98	1653.31
500	8529.53	8529.71	8580.29	8580.47
1000	16906.6	16906.7	16957.1	16957.3
5000	84486.8	84486.8	84537.2	84537.3
10000	169641	169641	169692	169692
15000	253294	253294	253344	253344
20000	337218	337218	337268	337268
25000	423542	423542	423593	423593
50000	845141	845141	845191	845191

TABLE V. AVERAGED OUTPUT OVER FIFTY RUNS



Processes	Avg WT		Avg TAT	
	SJF	K Factor	SJF	K Factor
10	149.286	152.944	198.854	202.512
50	819.932	821.54	870.028	871.636
100	1674.124	1675.178	1724.606	1725.662
500	8442.44	8442.7	8492.86	8493.12
1000	16970.3	16970.4	17020.84	17020.96
5000	84379	84379	84429.4	84429.4
10000	169143.4	169143.4	169194	169194
15000	253533.6	253533.7	253584.2	253584.2
20000	338352.1	338352.1	338402.6	338402.6
25000	423265.5	423265.5	423316.1	423316.1
50000	846220.8	846220.4	846271.1	846270.9

TABLE VI. AVERAGED OUTPUT OVER HUNDRED RUNS



Processes	Avg WT		Avg TAT	
	SJF	K Factor	SJF	K Factor
10	159.011	162.487	211.034	214.51
50	811.2964	813.1216	861.2442	863.0695
100	1670.259	1671.222	1720.603	1721.566
500	8387.21	8387.541	8437.523	8437.854
1000	16865.58	16865.74	16916	16916.16
5000	84384.3	84384.32	84434.75	84434.77
10000	168844.4	168844.4	168894.8	168894.8
15000	253529.5	253529.4	253579.8	253579.8
20000	337919.8	337919.8	337970.3	337970.3
25000	422741.9	422741.9	422792.4	422792.5
50000	845470.3	845470.3	845520.4	845520.4

As we can see in all of the above tables the k Factor may or may not give better results than SJF but it provides nearly the same effect but with a different selection of processes than SJF.

VII. CONCLUSION

This paper does not provide a better alternative for process scheduling than SJF, but it overcomes some of the limitations of SJF. As we know that, SJF gives less average waiting time for all available processes, it has more waiting time for processes which requires more time (burst time) to complete their execution and if the shorter processes arrive continuously then the starvation for long processes will be there.

In the above experiment, we demonstrated that the proposed algorithm removes the starvation of longer processes by becoming fair to all the process with similar turnaround time and waiting time as against SJF.

From the above experiment we can also conclude that as we increase the number of processes in the ready queue, the proposed algorithm starts to approximate more to the waiting time and turnaround time of SJF, even surpassing it in some cases.

REFERENCES

[1] Achyut S. Godbole, "Operating System with Case Studies in Unix, Netware, Windows NT", Tata McGraw Hill Publications.

[2] Silberschatz, Galvin," Operating System Concepts", Seventh Edition, John Wiley & Sons, Inc, 2005.

[3] Basit Shahzad,Muhammad Tanvir Afzal,"Optimized Solution to shortest job first by eliminating the starvation", Jordanian International Electrical Engineering And Electronic Conference (JIEEEC), 2005.

[4] Muhammad Akhtar, Bushra Hamid, Inayat ur-Rehman, Mamoona Humayun, Maryam Hamayun1 and Hira Khurshid ," An Optimized Shortest job first Scheduling Algorithm for CPU Scheduling", Journal of Applied Environmental and Biological Sciences,2015

[5] Jain, S., Shukla, D. and Jain, R. Linear Data Model Based Study of Improved Round Robin CPU Scheduling algorithm. International Journal of Advanced Research in Computer and Communication Engineering, June 2015, Vol. 4, No. 6, pp.562-56

[6] Andrew S. Tanenbaum, "Modern Operating Systems", Third Edition, Pearson Education International, 2009.

[7] Yadav, R., K., Mishra, A., K., Prakash, N. and Sharma, H. An improved Round Robin scheduling algorithm for CPU scheduling. International Journal on Computer Science and Engineering, 2010, No.02, pp. 1064-1066.

[8] Stalling, W. Operating System, Ed.5, New Delhi, Pearson Education, Singapore, Indian Edition, 2004.

[9] Tanenbaum, A. and Woodhull, A., S. Operating system, Ed. 8, New Delhi ,Prentice Hall of India, 2000.

[10] Dhamdhere, D., M. System Programming and Operating Systems, Revised.

[11] Shweta Jain, Dr. Saurabh Jain," A Review Study on the CPU Scheduling Algorithms", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 5, Issue 8, August 2016.