9th International scientific conference
Technics and Informatics in Education – TIE 2022
16-18 September 2022

Session: *Engineering Education and Practice*
Original scientific paper
DOI: 10.46793/TIE22.182T

Estimation of CPU Scheduling Algorithms Efficiency Using Object Oriented Programming

Milica Tufegdžić [1*], Vladeta Jevremović [1] and Zvonko Petrović [1]
[1] Academy of professional studies Šumadija/Department for Information technology, Trstenik, Serbia
[*] mtufegdzic@asss.edu.rs

Abstract: *Modern operating systems require sophisticated strategies for computer resources management that allows efficient allocation of Computer Processing Unit (CPU) within minimal response time for active users. CPU allocation shemas use different scheduling algorithms, such as First Come First Served (FCFS), Shortest Job First (SJF), Shortest Remaining Time (SRTF), and Round Robin (RR). These algorithms will be presented and evaluated in the terms of scheduling parameters, such as average waiting time (AWT), average turnaround time (ATT), average reponse time (ART), CPU utilization and throughput. Scheduling criteria metrics is done over two sets of input data, within 20 cases, with five processes in each case. Cases in which processes have the same arrival times (ATs) and cases when processes arrive in different times are considered. Randomly generated ATs and burst times (BTs) are used as input data in executable files, obtained from C++ source codes. Files are running by clicking on proper button from Graphical User Interface (GUI), developed in Python programming language. The results for obtained values of AWT, ATT, ART, CPU utilization and throughput for all cases and for FCFS, SJF, SRTF, and RR algorithms are analyzed and compared with the aim to estimate the efficiency of proposed algorithms.*

Keywords: *CPU scheduler; FCFS; SJF; SRTF; RR*

1. INTRODUCTION

Computer resources sharing is one of the basic tasks of a modern multitasking and multiprogramming operating system (OS). Managing and allocation of Central Processing Unit (CPU) between programs in executions, known as processes, is done by the part of OS called CPU scheduler. Whenever the CPU becomes idle, CPU scheduler makes a decision about which process should run at a certain point in time by selecting one of them for execution from the ready queue [1]. Three types of schedulers perform mediation in access to CPU: long-term scheduler, medium-term scheduler and short-term scheduler [2,3,4,5]. Long-term scheduler or admission scheduler decides which processes are to be accepted to the ready queue. Medium-term or mid-term scheduler temporarily removes processes from primary memory to secondary memory or vice-versa. Short-time scheduler, also known as dispatcher or CPU scheduler, decides which of the processes from the ready queue are to be allocated to CPU [2,3,4].

There are two main policies for switching CPU between multi processes, classified into preemptive and non-preemptive scheduling [1,4,6]. In preemptive scheduling, the process may be interrupted on force for some time, and the CPU will be allocated to another process, due to certain critical command [2,5,7]. The interrupted process will be resumed when the process of higher priority finishes its' execution [6,7]. On the contrary, there is no interruption of running process until it terminates in the case of non-preemptive scheduler [2,5,6,7]. This scheduler is also known as "voluntary" or "co-operative" [2,3].

Making decision about assigning a CPU core to a specific process from the ready queue is based on several algorithms: First Come First Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Priority Scheduling (PS), Round Robin (RR), Multilevel Queue Scheduling (MQS), Multilevel Feedback Queue (MFQ), and Completely Fair Scheduler (CFS) [1,2,4,5,8,9]. Some algorithms as the opposite of SJF and SRTF are proposed, such as the Longest job first scheduling non-preemptive and the Longest remaining time first scheduling [4]. MQS proposed dividing ready queues into several separate queues, while MFQ or "front-ground/back-ground multilevel" allows the process to move between queues [5]. In the Highest Response Ratio Next scheduling algorithm the priority of the process was assigned to the process based on the response ratio which depends on process' waiting time (WT) and service time. Different approaches were used for calculating the process priority in varying response ratio priority scheduling technique [10]. CFS, as the default Linux scheduling algorithm from release 2.6.23 of

the kernel, is based on scheduling classes, with each runnable task placed in a red-black tree [1].

Scheduling algorithms comparison is based on multiple criteria, such as CPU utilization, throughput, WT, turnaround time (TT), response time (RT), and fairness [1,2,3,4,6,9,10,11]. Some authors suggest additional criteria, such as context switch [6,9,10]. CPU utilization determines the percentage of time the processor is busy [1,2,6]. The number of processes that are fully executed for unit of time represents throughput [1,6,9,10,11]. The total time that the process spends waiting in the ready queue is WT [1,2,6,9,10], while TT represents the difference between time of completion and time of process submission [2,3,6,9,10]. The period from process submission to first reaction production is RT [1,2,3,4,6,10,11]. Given the equal opportunity to all processes to execute and prevent starvation is fairness [2]. Every time when a process in execution is interrupted before it is finished, with the aim to allocate CPU to another process, a context switch occurs [6,9,10].

For the purpose of this study, two sets of 10 cases with five processes in each case are determined. In order to evaluate FCFS, SJF, SRTF and RR scheduling algorithms, the executable files, converted from source codes in C++ programming language, are used. These executable files are running from proper GUI that was developed for this purpose in Python programming language. The results in the form of values for scheduling parameters are analyzed and estimation the efficiency for chosen algorithms is conducted.

2. SCHEDULING ALGORITHMS

In the simple FCFS algorithm, based on the process's arrival to the queue, the process which enters to the ready queue first is executed first [1,2,5,8,9]. The implementation is managed with FIFO queue [1,4]. When the CPU is assigned to the process, there is no interruption until it is terminated, which classifies FCFS into the class of non-preemptive schedulers [2,5,9]. The average waiting time (AWT) and average turnaround time (ATT) for high priority processes are often quite long, which are the main disadvantages of FCFS [1,9]. The order of processes' execution is according to their CPU BT in SJF algorithm, which means that the process with the least CPU BT will be executed first [5,8,9]. In the case when two process have the same BTs, FCFS algorithm is used for CPU allocation [1]. SJF algorithm can be implemented as non-preemptive (described above) and preemptive, known as SRTF [1,2,4,9]. The process that is currently executed will be preempted when newly arrived process has less CPU BT in SRTF implementation [1]. Both SJF algorithms have better performances than FCFS algorithm, considering AWTs and ATTs [4]. In PS priority is associated with each process and the processes which have the higher priority are executed first [1,2,8]. If two processes have the same priority, FCFS scheduling is implemented for CPU allocation [1]. Some issues, like indefinite blocking (starvation) of the processes with less priority, may arise in PS implementation [5,9]. Another type of preemptive scheduling that is widely used is RR scheduling, designed for time-sharing systems [4,8,9]. It is similar to FCFS scheduling, but added preemption enables switching between processes from circular ready queue [1]. Execution of every process is done within particular time quantum or time slice, usually from 10 to 100 milliseconds [1,5,9]. After that, CPU is allocated to a new process, while the old process is returned to the ready queue, waiting for its' turn again [4,8]. RR algorithm provides an average share of time for every process, but the performances mostly depend on the size of time quantum [1,4].

The AWT, the ATT, and the number of context switches for five processes with different arrival times were calculated and compared for FCFS, SJF, SRTF, PS, and RR scheduling techniques [9]. Five different cases for four processes P_1, P_2, P_3 and P_4 with different BTs and priorities were scheduled using FCFS, SJF, RR and PS algorithms. The values of AWTs and ATTs were calculated with the aim to compare all algorithms. The results of the study indicated that for every case the SJF algorithm was the most efficient, taking into account AWT and ATT, due to its' lowest values [8]. FCFS, SJF, RR and PS scheduling were compared in accordance to calculated values for TT and WT for each of the processes, as well as to ATT and AWT for every scheduling algorithm. These led to conclusion that FCFS and SJF are suitable for batch OS, while RR and PS are the most suitable for time-sharing systems [3]. Modeling of FCFS, Last-Come First-Served and SJF scheduling policies was done with neural networks [6].

Some variants of RR algorithms like shortest job RR, enhanced RR and adaptive RR were introduced with the aim to improve WT, AWT, TT, and ATT for RR scheduling [12]. RR scheduling with time quantum equal to the shortest BT of all processes solved the problem of starvation and enhanced performances which are expressed through reduction of ATT and AWT [11]. Modified RR with Dynamic Time Quantum used time quantum that was equivalent to average value of BT of the remaining processes in ready queue, rounded to the nearest integer value. Time quantum was updated after each cycle. This modified RR algorithm reduced the AWT, ATT, and number of context switches, comparing with traditional RR [13].

The processes were classified in two separate queues, one for processes with higher priority, and

the other for processes with lower priorities. The processes were alternatively chosen, starting from the process from queue with higher priority. This method reduced by half the AWT for high priority processes [7].

Maximizing throughput and CPU utilization, minimizing TT, RT, and WT, as well as the minimum overhead of context switches are the basic goals of every scheduling algorithm [4,5,6,9,10]. Choosing the proper scheduling algorithm relies on basic criteria metrics.

3. RESEARCH METHODOLOGY

The first step in this study was obtaining two input data sets. The first one consists of 10 cases with all processes which arrive at the same time, and the second, in which the processes in every case have different arrival times. All cases have five processes, namely P_1, P_2, P_3, P_4, and P_5. The values for BTs for every case and every process in the first set are generated using random.sample() method, for the range of 1-20, from random module. The values for ATs and BTs for each process from the second set are generated using random.randint() method, for the range of 0-20, from numpy module in Python programming language. The sets of input data are presented in Tables 1 and 2.

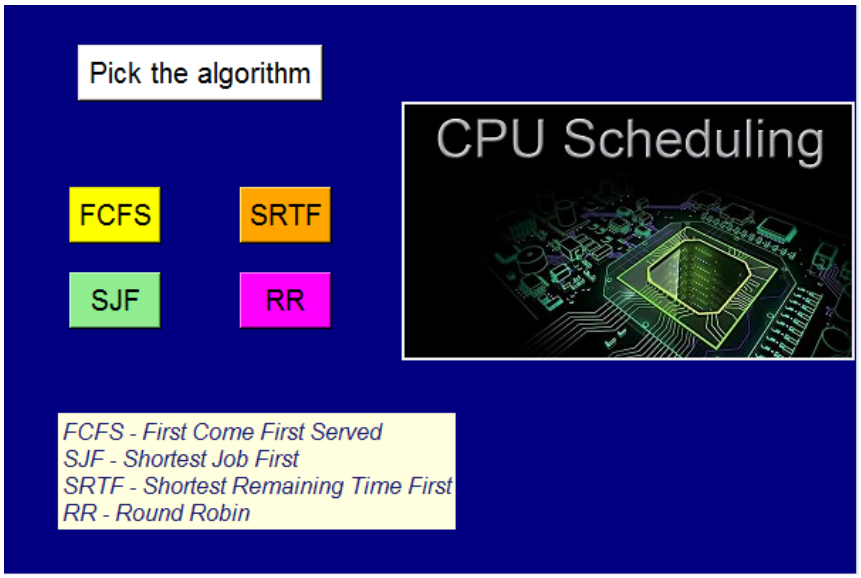
Source codes written in C++ programming language for FCFS, SJF, SRTF, and RR scheduling algorithms [14], are adapted and converted to executable files. GUI with labels in Python programming language is designed (Fig. 1) with the aid of tkinter module and its' methods.

Table 1. *Processes' burst times for the first set*

Process	Burst times (ms)				
	P_1	P_2	P_3	P_4	P_5
Case 1	1	6	8	3	16
Case 2	3	4	16	8	5
Case 3	18	14	13	15	7
Case 4	9	16	18	12	1
Case 5	12	15	17	4	16
Case 6	7	10	1	18	15
Case 7	13	12	2	10	8
Case 8	8	1	12	17	2
Case 9	15	1	9	11	16
Case 10	10	8	6	16	12

Table 2. *Process' arrival times and burst times for the second set*

Process	Arrival times and Burst times (ms)									
	P_1		P_2		P_3		P_4		P_5	
	AT	BT	AT	BT	AT	BT	AT	BT	AT	BT
Case 1	7	9	8	15	6	14	13	18	3	4
Case 2	6	10	1	15	0	2	7	17	1	1
Case 3	3	18	5	9	13	16	4	16	13	18
Case 4	3	10	2	18	8	9	2	10	6	7
Case 5	10	15	0	9	2	3	3	15	5	6
Case 6	0	9	3	10	10	16	8	10	6	10
Case 7	7	12	1	7	11	16	15	16	8	16
Case 8	5	18	6	13	2	17	8	14	2	7
Case 9	6	17	0	17	14	19	6	17	14	17
Case 10	1	10	1	17	17	18	14	15	10	16



Simple click on the button named as appropriate algorithm (see Fig. 1) allows for input data entry, AT, BT, and Time quantum for RR algorithm. The values such as Start time of the process (ST), Completion time of the process (CT), TT, WT, and RT, are obtained. TT, WT and RT are calculated using the equations (1) - (3).

$$TT = CT - AT \quad (1)$$

$$WT = TT - BT \quad (2)$$

$$RT = ST - AT \quad (3)$$

The time quantum for RR is set to 2 ms. Cases are tested on Windows 11 with Intel i5 12600k and 16 GB of RAM. The results of execution for FCFS algorithm are presented in Fig.2, as an example.

```
P No.    AT       BT       ST       CT       TT       WT       RT
1        0        1        0        1        1        0        0
2        0        6        1        7        7        1        1
3        0        8        7        15       15       7        7
4        0        3        15       18       18       15       15
5        0        16       18       34       34       18       18

Average Turnaround Time = 15.00
Average Waiting Time = 8.20
Average Response Time = 8.20
CPU Utilization = 100.00%
Throughput = 0.15 process/unit time
Press any key to continue . . .
```

Figure 2. *Results of the execution in the case of FCFS algorithm*

4. RESULTS AND DISCUSSION

The values for ART are excluded due the fact that these values are equal to AWT in all cases for FCFS, SJF and SRTF algorithms. There are differences for RR algorithm, where ART is less than AWT.

Table 3. *AWT and ATT for the first set*

	FCFS		SJF		SRTF		RR	
	AWT	ATT	AWT	ATT	AWT	ATT	AWT	ATT
Case 1	8.2	15.0	6.6	13.4	6.60	13.4	11.4	18.2
Case 2	12.8	20.0	8.40	15.6	8.40	15.6	15.2	22.4
Case 3	31.0	44.4	31.0	44.4	22.0	35.4	44.6	58.0
Case 4	26.4	37.6	14.2	25.4	14.2	25.4	28.2	39.4
Case 5	26.2	39.0	11.0	20.8	19.6	32.4	37.2	50.0
Case 6	15.6	25.0	12.0	22.2	12.0	22.2	22.8	33.0
Case 7	20.4	29.4	12.8	21.8	12.8	21.8	24.8	33.8
Case 8	15.2	23.2	7.60	15.6	11.6	19.6	13.6	21.0
Case 9	18.4	28.8	13.6	24.0	13.6	24.0	26.4	36.8
Case 10	18.4	28.8	16.0	26.4	16.0	26.4	29.6	40.0

With the aim to compare and conduct evaluation of presented algorithms, the results of the study for the first set are shown in Figures 3 and 4, in the form of graphics for AWT and ATT, respectively. ART is not presented due the fact that the values are equal to AWT in all cases for FCFS, SJF and SRTF algorithm.

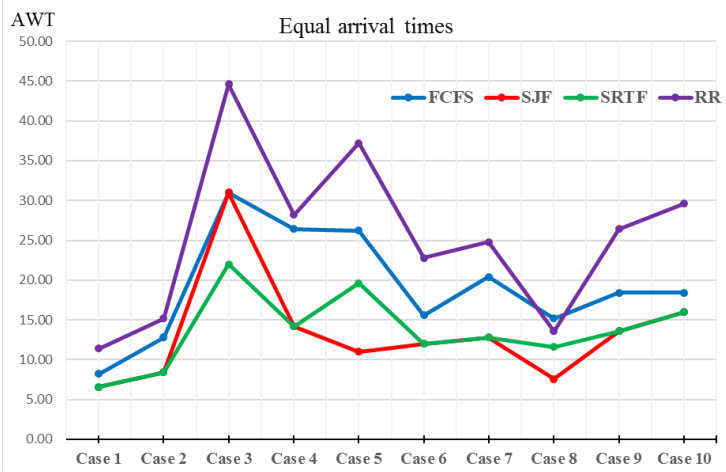


Figure 3. *The values of AWT for FCFS, SJF, SRTF and RR algorithms for the first set*

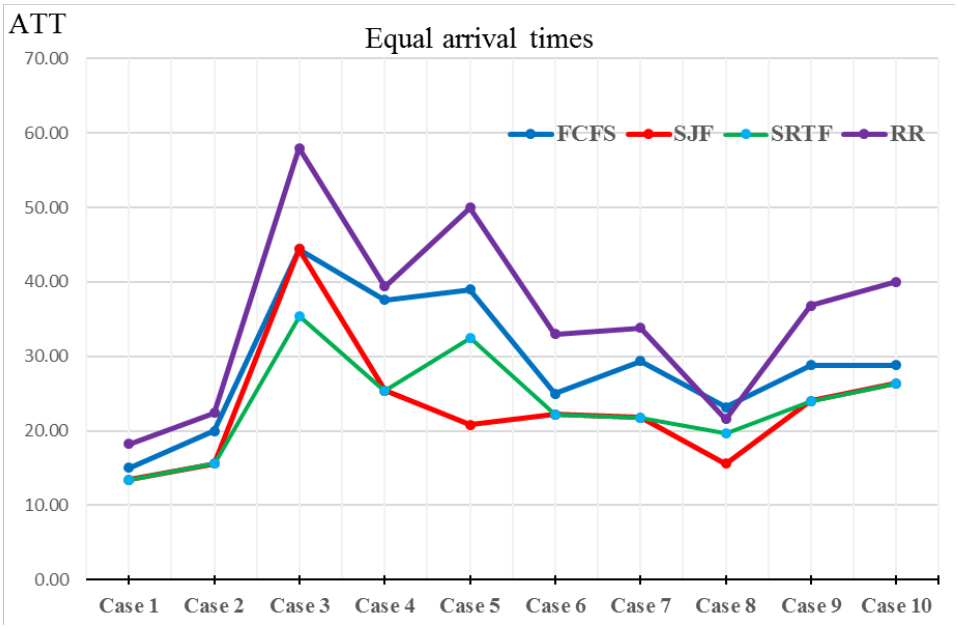


Figure 4. *The values of ATT for FCFS, SJF, SRTF and RR algorithms for the first set*

In general, SRTF and SJF algorithm gives the best performances according to AWT and ATT for all cases from the sets, compared to the other observed algorithms. This is especially the case when burst times are longer (Cases 3, 5, and 10). RR algorithm shows the worst performance, while SJF algorithm proved to be the best solution for CPU allocation according to the criteria of minimizing AWT and ATT, in the cases when all processes arrive at the same time.

Calculated values for the cases from the second set, for AWT and ATT are presented in Table 4, when all processes have different arrival times. The values for ART are excluded due to the fact that these values are equal to AWT in all cases for FCFS, SJF and SRTF algorithms. There are differences for RR algorithm, where ART is less than AWT.

Table 4. *AWT and ATT for the second set*

	FCFS		SJF		SRTF		RR	
	AWT	ATT	AWT	ATT	AWT	ATT	AWT	ATT
Case 1	13.8	28.8	12.8	24.8	12.8	24.8	23.6	35.6
Case 2	10.0	19.0	7.2	16.2	6.8	15.8	14.2	23.2
Case 3	26.2	41.6	24.8	40.2	23.0	38.4	44.2	59.6
Case 4	23.6	34.4	15.6	26.4	15.6	26.4	29.6	40.4
Case 5	12.2	21.8	10.4	20.0	9.40	19.0	21.4	31.0
Case 6	13.8	24.8	13.8	24.8	13.80	24.8	26.6	37.6
Case 7	15.0	28.4	15.0	28.4	15.0	28.4	27.8	41.2
Case 8	25.0	38.8	19.8	33.6	19.8	33.6	41.2	55.0
Case 9	26.4	43.8	26.0	43.4	26.0	43.4	53.2	70.6
Case 10	20.0	35.2	19.4	34.6	19.4	34.6	38.6	53.8

The results for the second set are shown in Figures 5 and 6, in the form of graphics for AWT and ATT, respectively. ART is not presented due to the fact that the values are equal to AWT in all cases for FCFS, SJF and SRTF algorithm.

SRTF algorithm gives the best performances according to the criteria of minimal AWT and ATT for all cases in which the processes have different arrival times. SJF is only slightly less effective then SRTF (just in two cases), while RR algorithm shows the worst performances. This is especially the case when BTs are longer than BTs from the other cases, as it is in Cases 3, and 9.

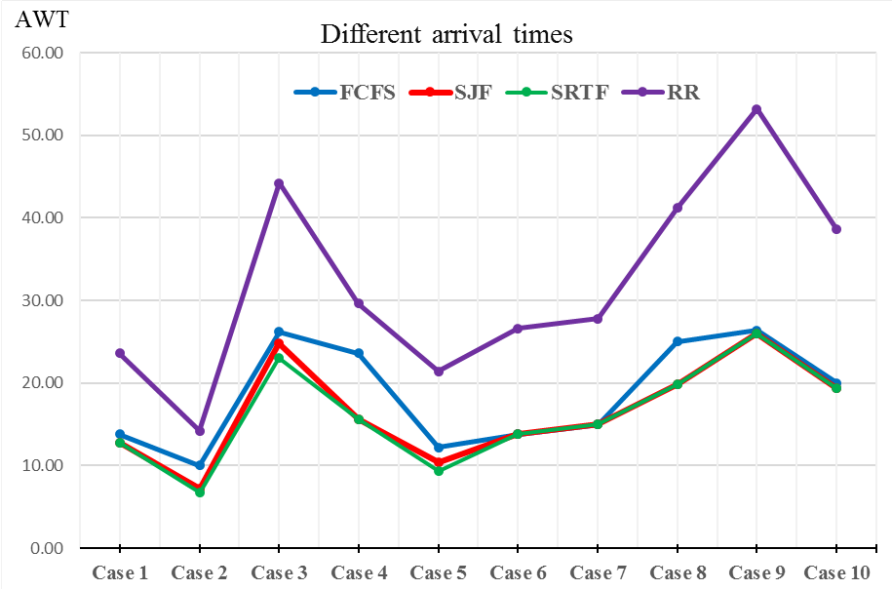


Figure 5. *The values of AWT for FCFS, SJF, SRTF and RR algorithms for the second set*

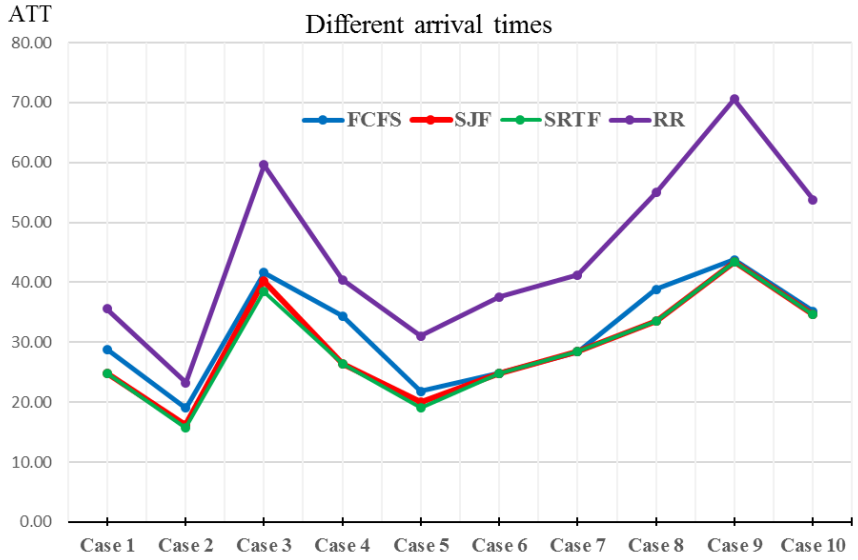


Figure 6. *The values of ATT for FCFS, SJF, SRTF and RR algorithms for the second set*

The values for CPU utilization is 100% in all cases from the first set for all algorithms. In 50% of the presented cases from the first set, the values for throughput are the same for all algorithms (Case 1 - 0.15, Case 5 - 0.08, Cases 6, 9 and 10 - 0.10). In the Cases 2, 3, 4, 7, and 8 these values are the same for FCFS, SJF and SRTF algorithms, respectively 0.14, 0.07, 0.09, 0.11 and 0.12. In these Cases the values for throughput are greater for RR algorithm (0.21, 0.13, 0.56, 0.15 and 0.56). In the second set the values for CPU utilization slightly deviates under 100% (see Table 5). In Cases 3, 9, and 10 there is no difference between the values for CPU utilization and throughput for all observed algorithms. In cases 1, 2, 4 - 8 the values of throughput for RR are slightly higher than values for FCFS, SJF and SRTF algorithms.

Table 5. *CPU Utilization (CPU u) and Throughput (Thr) for the second set*

	FCFS, SJF, SRTF		RR	
	CPU u	Thr	CPU u	Thr
Case 1	95.24	0.08	100	0.19
Case 2	100	0.11	100	0.38
Case 3	96.25	0.06	96.25	0.06
Case 4	96.43	0.09	100	0.12
Case 5	100	0.10	100	0.15
Case 6	100	0.09	100	0.10
Case 7	98.53	0.07	100	0.08
Case 8	97.18	0.07	100	0.12
Case 9	100	0.06	100	0.06
Case 10	98.70	0.07	98.70	0.07

5. CONCLUSION

Efficient CPU management is a very important issue, especially in modern multitasking and multiprogramming operating systems, where CPU is constantly being allocated and deallocated many times to different processes during programs' execution. Another additional issue is increasing scheduling performances, through minimizing TT, WT and RT, with maximizing throughput and CPU utilization.

In this study, different issues of FCFS, SJF, SRTF, and RR algorithms, as examples of preemptive and non-preemptive scheduling are compared and evaluated in terms of AWT, ATT, ART, CPU utilization and throughput. The results show that the SRTF algorithm, followed by SJF algorithm, provides the best performances in the sense of AWT and ATT, while the worst is, according to these two criteria, RR algorithm. On the other side, RR algorithm proved to be the best according to the ART. CPU utilization is almost equal for all presented cases, while the value of throughput is is slightly better for RR algorithm in regard to other proposed algorithms.

The implementation of proposed methodology for scheduling algorithms' application is easy, due to a simple GUI. Further analysis can include some improvements in the future, in the sense of increasing the number of cases in both sets, and expanding the list of algorithms with priority scheduling algorithm. There is also a need to analyze the RR algorithm with different quantum values, with the aim to find the best solution in terms of AWT and ATT.

REFERENCES

[1] Silberschatz, A., Gagne, G. & Galvin, P.B. (2018). *Operating System Concepts*. Tenth Edition, Chapter 5, Wiley.

[2] Singh, A. (2016). Review and analysis of CPU scheduling algorithms. *Biz and Bytes, 7(1),* 148-152.

[3] Goel, N. & Garg, R.B. (2012, November). A comparative study of CPU scheduling algorithms. *International Journal of Graphics & Image Processing*, *2(4),* 245-251.

[4] Omar, H.K., Kamal H. Jihad,K.H. & Hussein, S.F. (October 2021). Comparative analysis of the essential CPU scheduling algorithms. *Bulletin of Electrical Engineering and Informatics*, *10(5),* 2742-2750. doi:10.11591/eei.v10i5.2812

[5] Naji A. Majedkan, N., Ahmed, A.J., & Haji, L.M. (2020). CPU scheduling techniques: a review on novel approaches strategy and performance assessment. *Journal of Applied Science and Technology Trends, 01(02),* 48 – 55.

[6] Senan, S. (2017, Temmuz). A neural net-based approach for CPU utilization. *Bilişim teknolojileri dergisi*, *10(3),* 263-272.

[7] Arya, G.P., Nilay, K. & Prasad, D. (2018). An improved round robin CPU scheduling algorithm based on priority of process. *International Journal of Engineering & Technology, 7(4.5),* 238-241.

[8] Biswarup S. & Arinjay B. (2021). A detail analysis on AWT and ATT for different CPU scheduling algorithms for different cases. *Turkish Journal of Computer and Mathematics Education, 12(13),* 6994-7000

[9] Gupta, A.K., Yadav N.S. & and Dinesh Goyal, D. (2017, July). Comparative study of CPU scheduling algorithms. *International Journal of Recent Scientific Research, 8(7), 18851-18854.* doi:10.24327/IJRSR

[10] Singh, P., Pandey, A. & Mekonnen, A. (2015). Varying response ratio priority: a preemptive CPU scheduling algorithm (VRRP). *Journal of Computer and Communications, 3,* 40-51. doi:10.4236/jcc.2015.34005

[11] Chandiramani, K., Verma, R. & Sivagami, M. (2019). A modified priority preemptive algorithm for CPU scheduling. *Procedia Computer Science*, *165,* 363–369.

[12] Parashar, M. & Amit Chugh, A. (2014, July). Time quantum based CPU scheduling algorithm. *International Journal of Computer Applications, 98(3),* 45-48.

[13] Sohrawordi, E.A., Uddin, P. & Mahabub Hossain, M. (2019). A modified round robin CPU scheduling algorithm with dynamic time quantum. *Int. J. Adv. Res., 7(2),* 422-429. doi:10.21474/IJAR01/8506

[14] https://github.com/codophobia/process-scheduling-algorithms, accessed 12.06.2022.