Developed Dynamic Multilevel Feedback Scheduling Algorithm with an Aging Criteria for Starvation Mitigation

Nader AbdAlGhani* Mahmoud Mohamad†, Muhanad Atef‡ and Khaled Amgad§
Computer Engineering Dept., Faculty of Engineering, Cairo University
Cairo, Egypt
*nader_abdelghani@hotmail.com, †mmmacmp@gmail.com, ‡muhanad.atef23@gmail.com, §khaled.mohamed98@eng-st.cu.edu.eg

***Abstract*—Several scheduling algorithms are discussed in this paper with a certain objective in mind, to improve the performance of the Multilevel Feedback Queue (MLFQ) algorithm including the Round Robin (RR) algorithm as the latter is used to schedule the processes present in the queues of the former. Just as any scheduling algorithm, the standard MLFQ algorithm comes with its drawbacks which stem from the methodology used to calculate each queue quantum time value and the policies used to decide whether a process should be promoted or have its priority decreased. We have introduced an enhanced version of the MLFQ algorithm which computes each queue quantum value by multiplying the median burst time value by a factor of the queue level number, hence the increase of quantum value as priority decreases. This proposed approach also guarantees priority boosting whenever a process waiting time exceeds a certain threshold relative to its remaining time. Simulation performance analysis shows promising results in terms of decreasing the average turnaround time and the average waiting time.**

***Index Terms*—Operating systems scheduling, Multilevel feedback queue scheduling, Average turnaround time, Average waiting time, Process Aging, Round-robin scheduling.**

I. Introduction

Scheduling is a crucial part of many real-time applications such as scheduling airlines and railways communications, product manufacturing processes, data packets routing in computer networks and pipe networks. Operating systems multiprocessing environments are no different than those mediums mentioned, where processes compete over CPU utilization. This brought to existence the need for scheduling algorithms to justly assign processes to available CPUs in favour of optimizing performance measures, in particular, to maximize overall CPU utilization and throughput, and to minimize response time, waiting time and turnaround time. A schedular could be preemptive where it can temporarily interrupt a process without its cooperation and assign the CPU resources to a different process with the intention to assign them back to the former process, such operation is called context switching, or it could be non-preemptive (cooperative) by not being able to context switch between processes. Various preemptive and non-preemptive scheduling disciplines exist such as, first come, first serve (FCFS), shortest job first (SJF), shortest remaining time first (SRTF), round-robin (RR), multilevel queue and multilevel feedback queue. In FCFS, processes are non-preemptively executed according to their arrival time. In SJF, the CPU is assigned to the process with the smallest burst time. SRTF is a preemptive version of SJF where at each iteration, the process with the least remaining burst time takes control of the CPU. Round-robin preemptively assigns the CPU to each process in the ready queue for a static amount of time called quantum and executes them in an FCFS fashion. Multilevel queue algorithm partitions the ready queue into several queues to which processes are perpetually assigned and are executed according to another scheduling algorithm (e.g. RR). Processes cannot move from one queue to another. However, in a multilevel feedback queue algorithm, processes that don't terminate in one queue, due to having CPU burst time more than the time quantum assigned to their particular queue, are shifted to a lower priority queue. Due to the fact that long processes eventually sink to the lowest priority queue, they are carried out using FCFS to prevent starvation. Several papers discuss different methods to optimize the algorithms mentioned, each come with their set of advantages and drawbacks. One of many is [2], whose approach achieves better average response time but at the cost of hindering the scheduling process due to recalculating time quantum for each queue using a recurrent neural network. Our proposed approach implements the best of all worlds regarding [1] and [3] by avoiding each drawback and optimizing their advantages.

II. Related Work

Several papers proposed various types of approaches to improve the overall efficiency of the multilevel feedback queue scheduling algorithm. The chosen quantum time for each queue plays a major role. For instance, in [1], the highest priority queue has been given a relatively low quantum period, and as the priority of a queue decreases, its quantum gets multiplied by a constant. Hence, it is essential to choose a proper method to compute the time quantum value to minimize response time and maximize overall performance. In [2], an algorithm is presented for solving these drawbacks and minimizing the response time. In this algorithm, a Recurrent Neural Network (RNN) was used to determine both the number of queues and the optimized time quantum value for each queue. The RNN generates an effective model to compute the time quantum value. Their proposed intelligent version of the MLFQ offers good results, however, it suffers from a few drawbacks, the first being the direct proportionality of its network learning time and the amount of input data, and the second is the possibility of experiencing initial overhead at the first iterations of the algorithm. Our approach proposes an enhanced version of MLFQ using an optimized version of RR named shortest remaining burst round-robin (SRBRR) as in [3] which avoids the learning and overhead time in [2]. Regarding the various approaches to improve the MLFQ algorithm, those attempts dealt with starvation by assigning different quantum values to the ready queues depending on their priority. Our approach deals with starvation by boosting processes from lower priority queues to higher ones according to a specific policy.

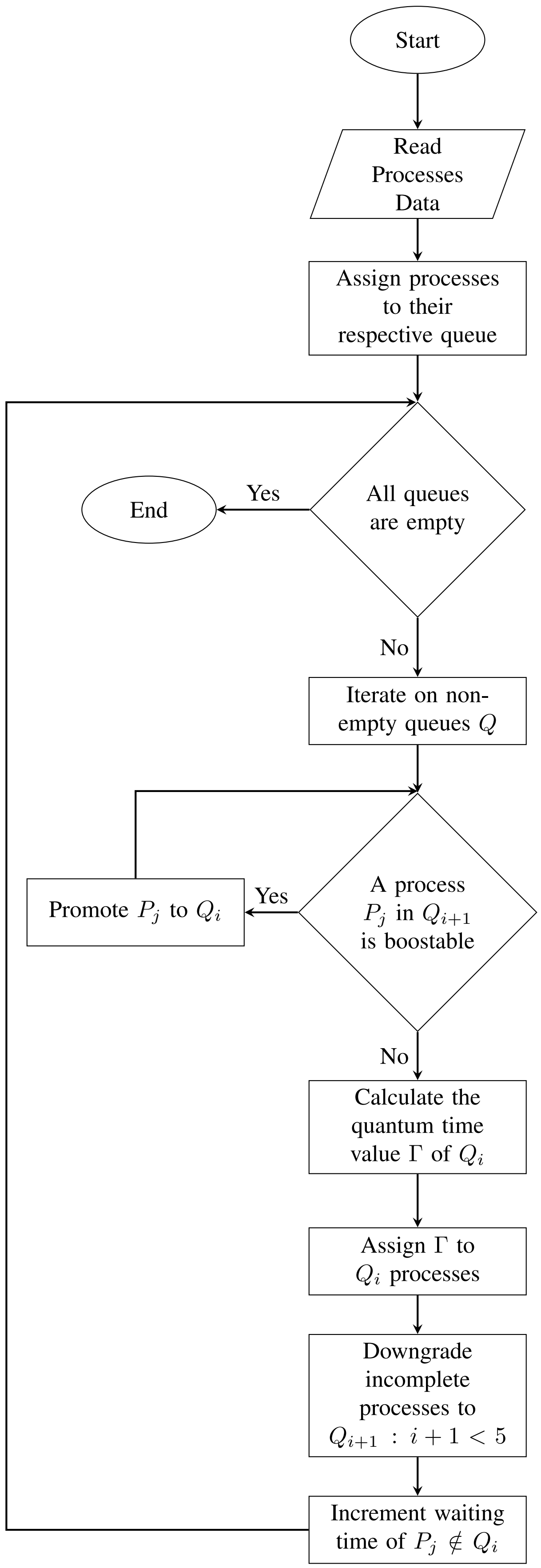
III. Proposed Approach

For a multilevel feedback queue scheduling algorithm, three parameters should be considered. The first being the chosen scheduling algorithm for each queue, especially the last queue as its scheduling algorithm must help treat starvation. The second is the criteria according to which a process priority should be increased based on its waiting time in its particular ready-queue, this technique is also known as Aging. The third is the criteria according to which a process priority should be decreased. In our proposed algorithm, there are 5 queues sorted in ascending order in line with their priority number, 0 being the highest priority and 4 being the lowest. Each queue uses a modified version of the round-robin scheduling algorithm stated in [3]. In [3], processes are sorted in ascending order according to their burst time and are assigned a time quantum that equals the median burst time of those processes. This dynamic quantum time RR algorithm provides better turnaround time and waiting time than the standard static quantum time RR algorithm whose quantum if set too short, leads to many context switches and if set too long, morphs the algorithm into an FCFS algorithm. The proposed alteration on the stated algorithm in [3] is that each queue quantum time isn't exactly the median burst time, but the burst time whose index is shifted from the median value towards the higher burst time values with a step count equal to its queue number, hence, the gradual increase of quantum time as priority decreases. For clarification, a queue with priority equal 2 has the following processes denoted by their burst time: 100, 300, 550, 600, 620, 700, 720, 900, 1200, the median value is 620, assuming we are using zero-based numbering, its index equals 4, and since we are in a queue whose priority equals 2, therefore the quantum slice value according to the proposed approach equals the burst value at index 6 (as in 4 + 2) which is 720. If the processes of a certain queue didn't terminate after assigning its queue quantum time, they are shifted downwards to the next lower priority queue. After introducing new processes into a queue, the quantum time slice is recalculated. Processes age whenever they are in a queue whose priority is one less than that currently getting scheduled and satisfies the following inequality:

$$1 - \frac{waiting\ time\ of\ P}{burst\ time\ of\ P} < 0.95 \tag{1}$$

Those procedures are repeated for all the generated queues until all processes reach the lowest priority queue where they are rescheduled until their completion.

A. Pseudocode Flowchart of the Proposed Approach



Start
Read Processes Data
Assign processes to their respective queue
All queues are empty
Yes
End
No
Iterate on non-empty queues Q
A process P_j in Q_{i+1} is boostable
Yes
Promote P_j to Q_i
No
Calculate the quantum time value Γ of Q_i
Assign Γ to Q_i processes
Downgrade incomplete processes to $Q_{i+1}\ :\ i+1 < 5$
Increment waiting time of $P_j \notin Q_i$

B. Proposed Algorithm

Algorithm 1 Enhanced Multilevel FeedBack Queue Scheduling Algorithm

```
 1: procedure EMLFQ(Number of processes $n$, priority of each process $\alpha_i$, burst time of each process $\beta_i$)
    Declaration and Initialisation:
 2:     Queue $Q_i$ where $i \in \{0,1,2,3,4\}$, waiting time $\omega_i = 0$ of $P_i$, quantum time $\Gamma$, quantum time index $\gamma$
 3:     for $i = 0$ to $n$ do
 4:         Assign $P_i$ to $Q_{\alpha_i}$
 5:     end for
    Loop Process:
 6:     $i = 0$
 7:     while $Q_i$ is NOT empty do
 8:         for $j = i + 1$ to $5$ do
 9:             if $1 - \omega_\theta/\beta_\theta < 0.95$ for $\theta \in Q_j$ then
10:                 Promote $P_\theta$ to the current queue $Q_i$
11:             end if
12:         end for
    Calculating the quantum time value:
13:         $ascending\_sort(\beta_{Q_i})$
14:         $\gamma = Q_i.size()/2 + i$
15:         $\Gamma = \beta_\gamma$ where $\beta_\gamma \in Q_i$
    Assigning the Quantum Value to each Process:
16:         for $P_j$ in $Q_i$ do
17:             $\beta_j = \beta_j - \Gamma$
18:             if $\beta_j <= 0$ then
19:                 Remove $P_j$ from $Q_i$
20:             else if $i < 4$ then
21:                 Downgrade $P_j$ to $Q_{i+1}$
22:             end if
23:         end for
    Increment Waiting Time:
24:         for $j = 0$ to $n$ and $P_j \notin Q_i$ do
25:             $\omega_j = \omega_j + 1$
26:         end for
    Increment Queue Index
27:         if $i < 4$ then
28:             $i = i + 1$
29:         end if
30:     end while
31: end procedure
```

IV. Experimental Analysis

A. Assumptions

The proposed scheduling algorithm is software replicated using Python scripts which simulate scheduling independent CPU-bound processes with predetermined burst time, arrival time and the queue to which they belong on a single processor environment which guarantees that no more than a single process uses the CPU at any arbitrary moment.

B. Experimental Scheme

On one hand, the input arguments to the proposed algorithm implementation are the number of processes to be scheduled, burst time, arrival time and the queue to which they belong to. On the other hand, output parameters are the average waiting time, average turnaround time and throughput. The following equations are used to calculate the previously mentioned output parameters:

$$Average\ Waiting\ Time = \frac{Total\ Waiting\ Time}{Number\ of\ Processes} \tag{2}$$

$$Average\ Turnaround\ Time = \frac{Total\ Turnaround\ Time}{Number\ of\ Processes} \tag{3}$$

$$Throughput = \frac{Number\ of\ Executed\ Processes}{Total\ Execution\ Time} \tag{4}$$

C. Performance Metrics

As a means to have a concrete viable evaluation of either the proposed algorithm or any other scheduling algorithm, the output parameters are taken into consideration for analysis. Since the average waiting time indicates the average time that a process has to starve for, therefore the lower the average waiting time is the better. The same principle applies to the average turnaround time and the number of context switches, as the former implies the average time spent by the process since its arrival time to its completion and the latter costs time as the CPU is assigned back and forth between different processes. Contrarily to the prior metrics, the larger the throughput is the better as it indicates the number of processes that are completely executed per unit time.

D. Simulation

For the sake of showcasing the proposed algorithm, a number of processes, their predetermined burst time and arrival time are taken as input to a Python simulation script. Suppose that the input to the script is according to the following table:

TABLE I

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	60	1
2	0	50	1
3	0	40	2
4	0	30	2
5	0	10	3
6	0	210	3
7	0	200	3

According to the proposed algorithm, the time quanta calculated are as follows:

TABLE II

Queue	*Quantum Value*
1	55
2	40
3	615
4	0
5	0

All processes are sorted ascendingly according to their remaining time scheduled in their respective queue through assigning the time quantum calculated for each queue. The time spent scheduling a particular queue is the waiting time for the subsequent queues.

The scheduling process goes as follows:

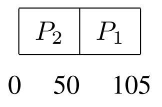


Fig. 1. Q_1 Gantt Chart

Considering that each process in Q_1 is assigned a quantum value of 50, as we reach the last process in Q_1, the total time consumed equals 105, which happens to be the time that all the other processes have to wait for in the subsequent queues hence the increase of their waiting time by 105 units of time.

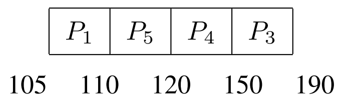


Fig. 2. Q_2 Gantt Chart

Even though P_5 is assigned to Q_3 as in Table XVI, it was promoted to Q_2 due to satisfying (1).

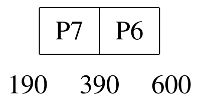


Fig. 3. Q_3 Gantt Chart

Whenever processes reach the lowest queue precompletion, they are scheduled using the RR scheduling algorithm with a relatively large quantum time value which is in most cases similar to using the FCFS algorithm because, as the time quantum value of the RR algorithm tends to infinity which could practically be a very large number relative to the available processes remaining time, the algorithm tends to morph into the FCFS algorithm. This procedure is repeated until all the processes are finished. Simulation results are shown in the table below:

TABLE III

Avg. Turnaround Time	*Avg. Waiting Time*	*Throughput*
230	144.3	0.011667

E. Performance Comparisons

To assess the performance of the proposed algorithm implementation, a varying number of processes is taken in six different experiments as input to the Python simulation script. In each experiment, the output of the proposed algorithm implementation is compared to the output of another scheduling algorithm implementation such as standard MLFQ with static quantum RR and other variants of MLFQ and RR addressed in different papers.

1) Experiment 1: In this experiment, the proposed algorithm is compared against an MLFQ algorithm stated in [4] that uses a static version of the RR algorithm for scheduling each queue and another variant that uses a dynamic version of the RR algorithm for doing so.

TABLE IV
EXPERIMENT 1 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	1	25	1
2	5	70	1
3	6	84	1
4	7	17	1
5	8	35	1

TABLE V
EXPERIMENT 1 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	115	68.8
Dynamic RR MLFQ	150.8	107.6
Static RR MLFQ	161.4	116.2

2) Experiment 2: In this experiment, the proposed algorithm is compared against an MLFQ algorithm stated in [4] that uses a static version of the SJFRR algorithm for scheduling each queue and another variant that uses a dynamic version of the SJFRR algorithm for doing so.

TABLE VI
EXPERIMENT 2 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	1	25	1
2	5	70	1
3	6	84	1
4	7	17	1
5	8	35	1

TABLE VII
EXPERIMENT 2 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	115	68.8
Dyn. SJFRR MLFQ	134	91.8
Stat. SJFRRMLFQ	143.4	98.2

3) Experiment 3: In this experiment, the proposed algorithm is compared against an MLFQ algorithm stated in [5] that uses a static version of the SJFRR algorithm for scheduling each queue and another variant that uses a dynamic version of the SJFRR algorithm for doing so.

TABLE VIII
EXPERIMENT 3 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	8	1
2	3	133	3
3	2	21	2
4	8	39	2
5	19	67	2
6	33	114	3
7	33	54	2

TABLE IX
EXPERIMENT 3 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	151	88.7
Dyn. SJFRR MLFQ	252	119
Stat. SJFRR MLFQ	351	228

4) Experiment 4: In this experiment, the proposed algorithm is compared against different variants of the MLFQ algorithm that are stated in [6]: standard MLFQ, a priority-based MLFQ and a vague logic-based MLFQ algorithm.

TABLE X
EXPERIMENT 4 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	40	1
2	0	30	1
3	0	50	1
4	2	70	1
5	4	25	1
6	6	60	1
7	7	45	1

TABLE XI
EXPERIMENT 4 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	185.85	140.14
VMLFQ	190	170
MLFQ	232.14	175
PMLFQ	240	180

5) Experiment 5: This experiment is the same as the previous one but with a different input test case.

TABLE XII
EXPERIMENT 5 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	90	1
2	0	30	1
3	0	28	1
4	0	57	1
5	0	73	1
6	0	19	1
7	0	42	1
8	0	67	1

TABLE XIII
EXPERIMENT 5 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	212.5	161.75
VMLFQ	260	225
MLFQ	290	240
PMLFQ	300	245

6) Experiment 6: In this experiment, the proposed algorithm is compared against two variants of the RR algorithm stated in [3]. The first is a static version of the RR algorithm with a constant quantum value of 25 for scheduling each queue while the second uses a dynamic version of the RR algorithm called SRBRR for doing so.

TABLE XIV
EXPERIMENT 6 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	13	1
2	0	35	1
3	0	46	1
4	0	63	1
5	0	97	1

TABLE XV
EXPERIMENT 6 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Proposed Algorithm	113.2	62.4
Dynamic SRBRR	122.4	71.6
Static RR	148.2	97.4

7) Experiment 7: This experiment is the same as the previous one but with a different input test case. Note that for this test case, a process queue number is irrelevant to both RR algorithm and SRBRR algorithm mentioned in [3].

TABLE XVI
EXPERIMENT 7 INPUT

Process	*Arrival Time*	*Burst Time*	*Queue*
1	0	54	1
2	0	99	3
3	0	5	2
4	0	27	2
5	0	32	2

TABLE XVII
EXPERIMENT 7 RESULTS

Algorithm	*Avg. Turnaround Time*	*Avg. Waiting Time*
Dynamic SRBRR	93.6	50.2
Proposed Algorithm	106.8	63.4
Static RR	152.2	108.8

F. Observation

From the above simulations of different test cases and multiple performance comparisons that involved as many as 11 different scheduling algorithms not including this paper algorithm, it is clear that the average turnaround time and the average waiting time of the proposed algorithm is less than or – in few occasions – nearly equal to those of the stated algorithms. With that said, the proposed algorithm is arguably advantageous over those algorithms, considering even the case in which it underperformed compared to the SRBRR algorithm, it is still favourable due to the capability to separate processes into categories based on their need for the processor and other advantages of the MLFQ algorithm.The performance of the proposed algorithm compared to other algorithms is further illustrated in the graphs below:

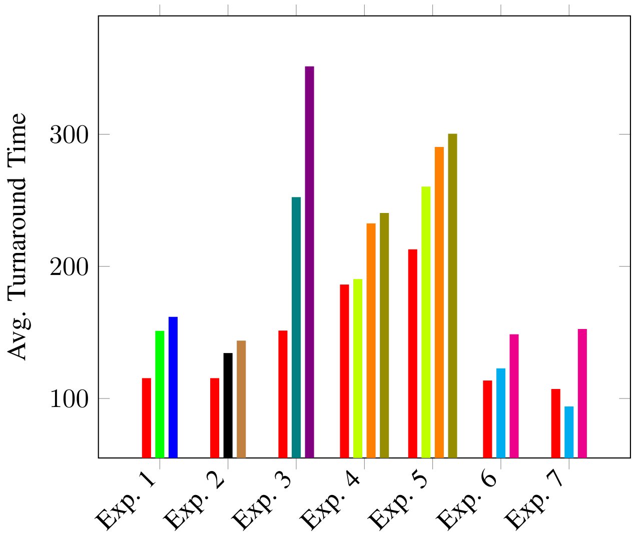


Fig. 4. Comparison graph for average turnaround time

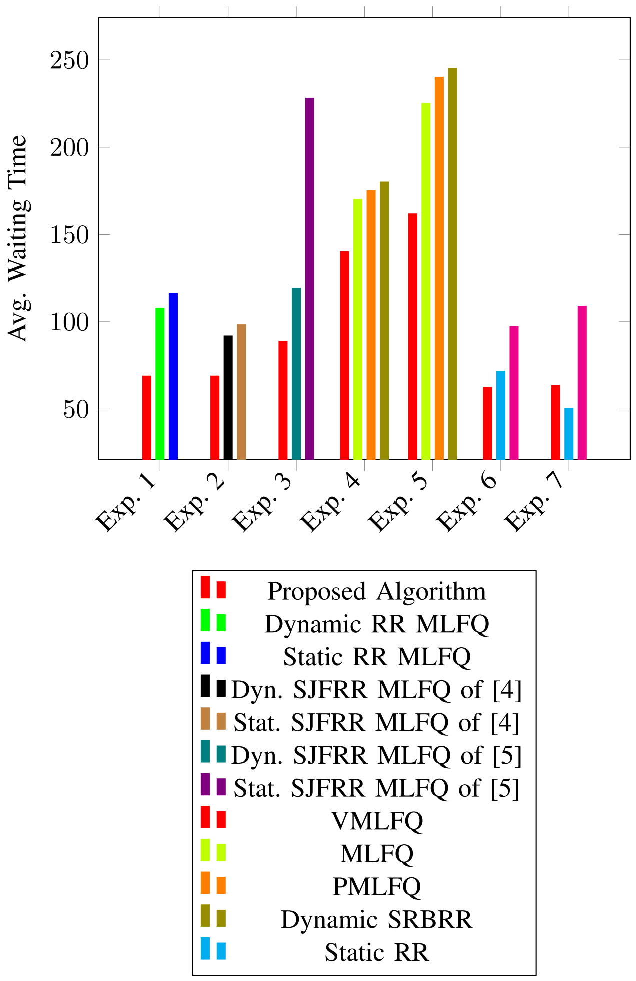


Fig. 5. Comparison graph for average waiting time

V. Conclusion and Future Work

The goal of this paper is to tackle different shortcomings associated with the standard MLFQ scheduling algorithm as well as its variants discussed in several papers. To resolve those deficiencies, we introduced different adjustable policies and techniques and it is evidently clear that those methods yield better CPU performance and optimize utilization by reducing the average waiting time as well as the average turnaround time. Despite experimenting numerous combinations of the parameters and scheduling policies by which an MLFQ algorithm operates, we can say that there is yet a large room for experiment and improvement through finding better methods and policies to make the algorithm more adaptable to the nature of the submitted processes and overall more enhanced.

References

[1] H.S. Behera, Reena Kumari Naik, Suchilagna Parida, “Improved multilevel feedback queue scheduling using dynamic time quantum and its performance analysis”, *International Journal of Computer Science and Information Technologies*, vol. 3, no. 2, 2012, pp. 3801-3807.

[2] MohammadReza EffatParvar, Karim Faez, Mehdi EffatParvar, Mehdi Zarei, Saeed Safari, “An intelligent MLFQ scheduling algorithm (IMLFQ) with fault tolerant mechanism”, *Sixth International Conference on Intelligent Systems Design and Applications*, vol. 3, 2006, pp. 80–85.

[3] Rakesh Mohanty, H. S. Behera, Khusbu Patwari, Monisha Dash, “Design and performance evaluation of a new proposed shortest remaining burst round robin (SRBRR) scheduling algorithm”, *International Symposium on Computer Engineering & Technology*, vol. 17, 2010.

[4] S. K. Dwivedi and R. Gupta, “A simulator based performance analysis of multilevel feedback queue scheduling”, *2014 International Conference on Computer and Communication Technology (ICCCT)*, Allahabad, 2014, pp. 341-346.

[5] Malhar Thombare, Rajiv Sukhwani, Priyam Shah, Sheetal Chaudhari, Pooja Raundale, “Efficient implementation of multilevel feedback queue scheduling”, *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Chennai, 2016, pp. 1950–1954.

[6] S. Raheja, R. Dadhich, and S. Rajpalc, “Designing of vague logic based multilevel feedback queue scheduler”, *Egyptian Informatics Journal*, vol. 17, 2016, pp. 125-137.

[7] A. Alsheikhy, R. Ammar and R. Elfouly, “An improved dynamic Round Robin scheduling algorithm based on a variant quantum time”, *2015 11th International Computer Engineering Conference (ICENCO)*, Cairo, 2015, pp. 98-104.

[8] M. K. Mishra and F. Rashid, “An improved round robin CPU scheduling algorithm with varying time quantum”, *International Journal of Computer Science, Engineering and Applications (IJCSEA)*, vol. 4, 2014.

[9] A. Singh, P. Goyal and S. Batra, “An optimized round robin scheduling algorithm for CPU scheduling”, *International Journal on Computer Science and Engineering*, 2010, pp. 2383-2385.

[10] Dipto Biswas, Md. Samsuddoha, “Determining proficient time quantum to improve the performance of round robin scheduling algorithm”, *International Journal of Modern Education and Computer Science(IJMECS)*, vol. 11, 2019, pp. 33-40.

[11] K. Hoganson and J. Brown, “Real-time scheduling with MLFQ-RT multilevel feedback queue with starvation mitigation”, *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, Funchal, 2017, pp. 155-160.