# An Efficient Dynamic Round Robin Algorithm for CPU scheduling

Muhammad Umar Farooq, Aamna Shakoor, Abu Bakar Siddique
University of Engineering and Technology, Lahore
mufarooq40@gmail.com

***Abstract*—The efficiency of embedded systems mainly depends on the process scheduling policy of operating systems running on them. Better scheduling algorithms make a system fast using less resources for small time. Out of some important scheduling algorithms, Round Robin algorithm is much efficient. But its efficiency fairly depends on choosen time quantum. In this paper, we have developed an efficient Round Robin algorithm using Dynamic Time Quantum. Some such systems have already been developed but they take advantage of other algorithms and their running time is much higher due to sorting of processes which is practically impossible. So, our goal is to reduce running time of an algorithm along with efficiency constraints such as context switches, average waiting and turnaround times. Lower the context switches, average waiting and turnaround times; higher the efficiency of an operating system and thus better embedded system. In the last section of this paper, we will present a comparison of our system with previously developed algorithms.**

***Index Terms*—Operating Systems (OS), CPU scheduling, Round Robin (RR) algorithm, Dyanmic time quantum**

## I. Introduction

CPU scheduling is one of the fundamental features that an Operating System (OS) needs to perform multitasking. CPU scheduling means allocation of sufficient resources to a process being executed. The policy of CPU scheduling is to maximize the efficiency of the CPU. This efficiency includes maximizing the throughput, minimizing context switches, Average Waiting Time (AWT) and Average Turnaround Time (ATT) of the processes in ready queue. Throughput can be defined as the number of processes executed per unit cycle. The process waiting time of a process is the time it spends in ready queue before its execution.Turn around time is the sum of waiting and the execution time of a process.

CPU scheduling algorithms are subdivided in to two main categories. A non-preemptive algorithm is the one in which CPU is allocated to a process till completion of its execution while in a preemptive algorithm, a higher priority process can block currently running process. Or we may simply conclude that if there are $n$ processes in a ready queue, a non-preemptive process can have only $n - 1$ context switches. While a preemptive process can have more context switches.

Some of the algorithms that have been developed and implemented are as follows;

*First Come First Serve (FCFS):* As obvious from the name, FCFS processes the jobs in the order in which they arrive in the ready queue. The floaw in this process is high average waiting and turnaround times when the shorter processes arive after the long processes.

*Shortest Job First (SJF):* In this algorithm, the ready queue is first sorted on ascending order of burst times of the processes arrived in it. Then the processes are assigned CPU in sequence. This algorithm is optimal than others in most cases. But in SJF, we must be able to have a pre-knowledge of burst times of all the processes which is difficult practically. Moreover, it can cause starvation for longer processes.

*Priority Scheduling (PS):* In priority scheduling, the processes come in ready queue along with assigned priorities to them. OS assigns CPU to the process with highest priority. Then the CPU is assigned to a process with lower priority and so on. Priority scheduling algorithm respects the priority of a process rather than focusing on efficiency constraints of CPU. So, it may cause best and worst cases depending on burst times of corresponding processes. FCFS and SJF, both are non-preemptive algorithms. But PS can be of both type i.e. preemptive or non-preemptive. If the currently executing process can continue its execution on arrival of a higher priority process, then it is a non-preemptive priority scheduling. On the other hand, if the process is blocked on arrival of a higher priority process, it is a preemptive priority scheduling.

*Round Robin (RR):* This algorithm allocates CPU to all processes for an equal time interval. A process is blocked and put at the end of ready queue after a constant time slice, known as Time Quantum. This process is assigned CPU time again once the execution of all other processes in their respective time quanta. The efficiency of Round Robin depends entirely on the time quantum selected. If the time quantum is selected too large, the algorithm will become FCFS. While on the other hand, if the quantum is much small, it will yield much overhead and larger average waiting and turnaround times [1].

With advancement in technology, many Dynamic Round Robin (DRR) algorithms have been developed. In DRR, a dynamic time quantum is chosen instead of a constant time quantum. It may be changed after a cycle or just after an arrival of next process in ready queue. Many CPU scheduling algorithms based on DRR have been developed but most of those algorithms take advantage of other algorithms (will be duscussed in detail in next section). But again efficiency of such algorithms depends on how a DRR time quantum is selected.

In this paper, we have contributed to develop a new DRR algorithm with higher efficiency and lesser context switches, less average waiting and turnaround times with high throughput. We will name our proposed algorithm as "Efficient Dynamic Round Robin (EDRR) algorithm" throughout this paper. The goal was to select a most optimal dyanmic time quantum to gain of advantage of RR algorithms. Comparison of our proposed algorithm with already developed algorithms is shown in the section of the paper to compare the efficiency and contraints of different processes.

## II. Literature Review

Now a days operating systems are capable of performing multitasking that depends on CPU scheduling algorithms tohandle simultaneous jobs. Many algorithms have been designed to increase the efficiency as much as possible. The basis of Round Robin (RR) algorithm, as discussed above, is on preemptive CPU scheduling with context switching after a regular time interval irrespective of process state. This time interval is termed as Time Quantum (TQ). In such an algorithm, the efficiency purely depends on the set time quantum. If the time quantum is set too high to burst times of processes in ready queue, the algorithm will become just FCFS algorithm. And if the time quantum is set too small, there will be more overhead for context switches and as a result increased waiting time for processes. The efficiency of RR is somewhere in middle. To maximize the efficiency of RR algorithm, the crucial thing is to select the time quantum cleverly. As a rule of thumb, RR algorithm produces a higher efficiency when a QT is selected greater than 80% of processes burst times [1]. To increase efficiency further, many extensions of RR algorithm with Dynamic Time Quantum (DTQ) has been developed. DTQ means to select a time quantum that may change during processing rather than choosing a constant time quantum. DTQ may be changed after one complete cycle or just after a new job added to ready queue.

The simplest approach to select a dynamic time quantum is to take mean of burst times of all available processes in ready queue [2]. The algorithm first sorts jobs on ascending order of their burst times and then processes them using RR algorithm with time quantum set equal to mean. However, the approach is too simple to increase efficiency significantly. The algorithm was named as Dynamic Average Burst time Round Robin (DABRR) algorithm. Another approach was introduced to set DQT equal to median of the available processes [3]. A threshold of 25 is defined to set time quantum. If the median of all available processes is less than the threshold, time quantum is set 25 otherwise it is set equal to the median. Time quantum is changed after each cycle based on median of all sorted available processes in ready queue. The algorithm was named as SARR. This approach was used by HS Behera et al [4] to improve context seitching. However, their proposed improvement (DQRRR) produced poor results in terms of average waiting and turnaround times.

All the described techniques update their quantum time after first execution cycle. A. Noon et al. update the quantum time right at a new process arrival in ready queue [5]. However, they use the same mean average technique to set quantum time. So still their algorithm produces more context switches and thus overhead. So, Ahmed Al shiekhy et al then described a more efficient algorithm [6]. They sort the processes on ascending order of their burst times and take average of largest two burst times. The quantum time is then set equal to this average. This algorithm produces best results among all described above. However, all the above described algorithms first sort processes on ascending order of their burst times [2], [3], [4], [6]. This tends their algorithm towards SJF and most of algorithms take advantage of optimization of SJF rather than their own algorithms. Moreover, the sorting on ascending order is itself of order $O(N^2)$. So, such algorithms face this problem as well along with some other problems of SJF due to inclination of their algorithms towards SJF.

So, in our proposed algorithm, we have tried to achieve best results in term of efficiency without sorting the processes on basis of their burst times. Then a comparison of our proposed algorithm with above described algorithms have been shown.

## III. Proposed Solution

As described earlier, the performance of Round Robin algorithm fairly depends on the chosen time quantum. A larger quantum leads to FCFS algorithm while a shorter quantum algorithm causes more context switches and thus a large overhead. So, the efficiency of Round Robin depends on how cleverly you select the time quantum. One of the problems with Round Robin algorithm with unclever time quantum is that sometimes a process has to wait for all other processes even if its remaining execution time is much smaller i.e. 1 or 2 time units. A simple problem is discussed below:

| | Burst Times | Waiting Times | Turnaround Times |
|---|---|---|---|
| P1 | 30 | 0+106=106 | 136 |
| P2 | 27 | 29 | 56 |
| P3 | 35 | 56+(136-85)=107 | 142 |
| P4 | 29 | 85 | 114 |
| P5 | 21 | 114 | 135 |
| **Average** | | 88.2 | 116.6 |

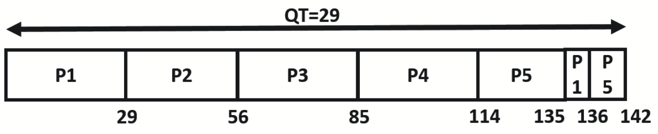

Fig. 1. Gantt chart of Problem with RR

In the example described above, P1 must wait for 106 milliseconds for the remaining just 1 millisecond execution time. If this 1 millisecond was also executed in starting, average waiting time could have been drastically reduced 67 milliseconds. Moreover, if the P3 was also executed completely first time, average waiting time and context switches can be further reduced.

Our approach was to select an elastic dynamic quantum that will allow a process to execute completely if its remaining execution time is less than or equal to 0.2th of its actual time. First, we find the maximum burst time from available processes in ready queue. Then take a proportion of this time to set the time quantum. As a rule of thumb, we have selected time quantum as 0.8th fraction of the maximum burst time. Now the scheduler assigns the CPU to all the processes in ready queue with burst time less than the time quantum while larger ones are kept hold on. As soon as all the smaller processes complete their execution, the time quantum is set equal to maximum burst time. This assignment covers our above described criteria to give enough time to all processes to complete. For different arrival times, the scheduler recalculates the time quantum at the end of current quantum only if a new process has been entered the ready queue.

Our proposed approach yields improved results without taking advantage of SJF and time wasting sorting of ready queue.

**Algorithm 1:** Pseduo Code for our proposed EDRR

```
$BT_{max}$=Maximum Burst Time;
$BT_i$=Burst Time of $i^{th}$ process;
$QT$=Quantum Time;
$N$=Number of processes in ready queue;
$remainingProcesses$=Remaining Processes;
$i = 1$;

$QT = 0.8 * BT_{max}$;
while $i <= N$ do
    if $i < N$ then
        if $B_i <= QT$ then
            assign CPU to the process;
            $N - -$;
        else if $B_i > QT$ then
            Dont assign CPU and put the process at the
              end of ready queue;
            $remainingProcesses + +$;
    else if $i == N \&\& remainingProcesses > 0$ then
        $QT = BT_{max}$;
        $i == 0$;
    $i + +$;
end
```

## IV. Experimental Analysis

Two examples are given below to understand the proposed algorithm. The context switches, average waiting times and average turnaround times have been calculated for each example and will be plotted against all other DRR algorithms in next section.

### A. Example 1: With same arrival time

The first example consists of five processes in ready queue. An asumption has been made here that all the processes arrived in ready queue at the same time. In other words, the arrival time of alll the processes is assumed the same.

| | Burst Times | Waiting Times | Turnaround Times |
|---|---|---|---|
| P1 | 80 | 141 | 221 |
| P2 | 45 | 0 | 45 |
| P3 | 62 | 45 | 107 |
| P4 | 34 | 107 | 141 |
| P5 | 78 | 221 | 299 |
| **Average** | | 102.8 | 162.6 |

According to our EDRR, initially the time quantum will be the $0.8$ times of $BT_{max}$. So, the first time quantum will be $0.8$ times 80 (the maximum burst time), that is 64. Now, burst times of all the processes will be compared with this value. CPU time will not be assigned to the process if the burst time is greater than this value and will be put at the end of the queue. On this criteria, P1 will not be assigned the CPU time as its burst time is greater than 64. However CPU will be assigned to P2,P3 and P4. Burst time of P5 is also greater than time quantum 64. After this first execution, according to EDRR the time quantum will become equal to $BT_{max}$ that is 80. So P1 and P5 will get CPU assigned now. The Gantt chart of the example is shown in Figure 2.

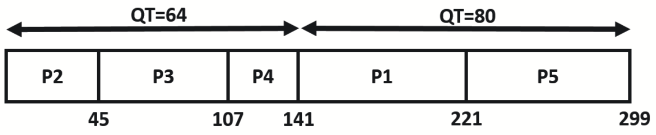

Fig. 2. Gantt chart of Example1

The waiting and arrival times, calculated from this gantt chart, is shown in above table.

### B. Example 2: With the different arrival times

Now discuss another example where all the processes arrive in ready queue with different arrival times. Again, we take five processes as an example, given in table;

| | Arrival Times | Burst Times | Waiting Times | Turnaround Times |
|---|---|---|---|---|
| P1 | 0 | 45 | 0 | 45 |
| P2 | 5 | 90 | 203 | 293 |
| P3 | 8 | 70 | 37 | 107 |
| P4 | 15 | 38 | 100 | 138 |
| P5 | 20 | 55 | 133 | 188 |
| **Average** | | | 94.6 | 154.2 |

According to our algorithm, initially only P1 is in the ready queue. So, the time quantum will be $0.8^{th}$ part of $BT_{P1}$. So, the first time quantum will be $0.8^{th}$ fraction of 45 that is 36. After 36 time units, all the processes will be there in ready queue. And now the time qunatum must be changed to $0.8^{th}$ part of $BT_{max}$. So, the time quantum will become $0.8^{th}$ fraction of 90 that is 72. So, now according to algorithm, remaining P1 will continue executing as it is lower than 72. Afterwards, P3,P4,P5 executes and P2 is put on the end of the queue that will execute on next time quantum equal to $BT_{max}$. The Gantt chart of the example is shown in Figure 9.

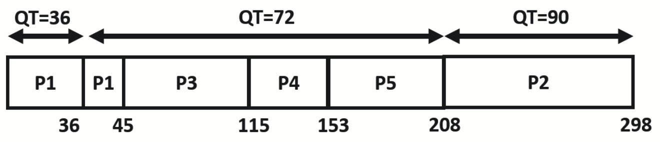

Fig. 3. Gantt chart of Example2

The waiting and arrival times, calculated from this gantt chart, is shown in above table.

## V. Comparison Analysis

Now we compare the results of our examples. For comaprison, lets take another example with same arrival times as described below;

| | Burst Times | Waiting Times | Turnaround Times |
|---|---|---|---|
| P1 | 40 | 0 | 40 |
| P2 | 55 | 40 | 95 |
| P3 | 60 | 95 | 155 |
| P4 | 90 | 155 | 245 |
| P5 | 102 | 245 | 347 |
| **Average** | | 107 | 176.4 |

The context switches, average waiting and turnaround times have been calculated according to EDRR decribed in previous section. Now compare the results with the algorithms described in literature review.

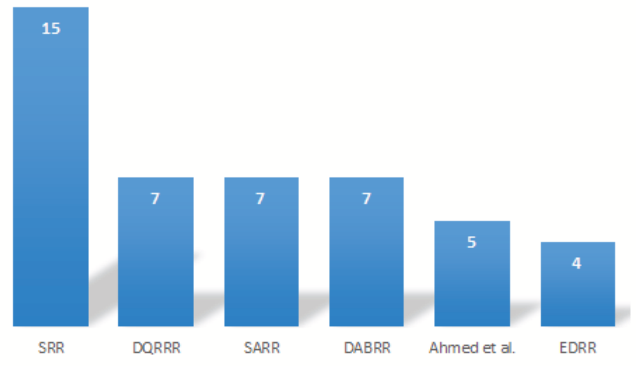

Fig. 4. Comparison of context switches

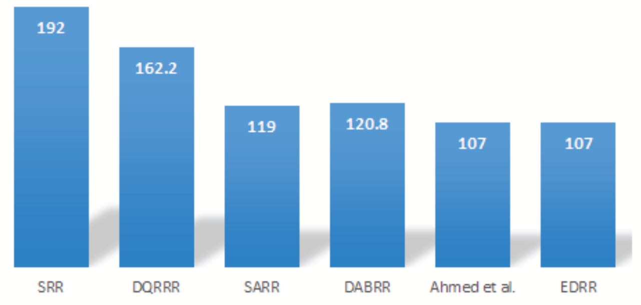

Fig. 5. Comparison of Average Waiting Time

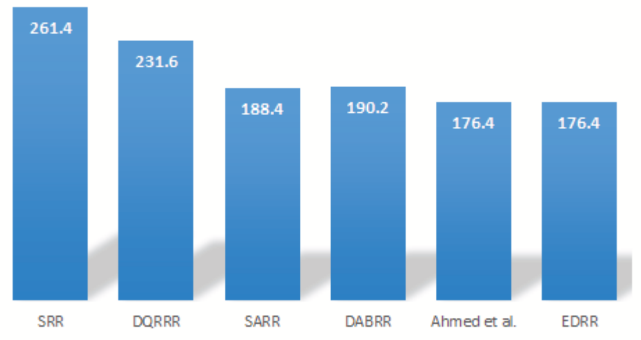

Fig. 6. Comparison of Average Turnaround Time

From above comparison, it is clear that our algorithm performs best in context switches scenario and is considerbly efficient in average waiting and turnaround times. Though our algorithm is not all time best in regards of average waiting and turnaround time compared with Ahmed et al. but it saves complexity as it doesn't take advantage of SJF and queue sorting in other algorithms.

For different arrival times, lets compare results of exapmle 2.

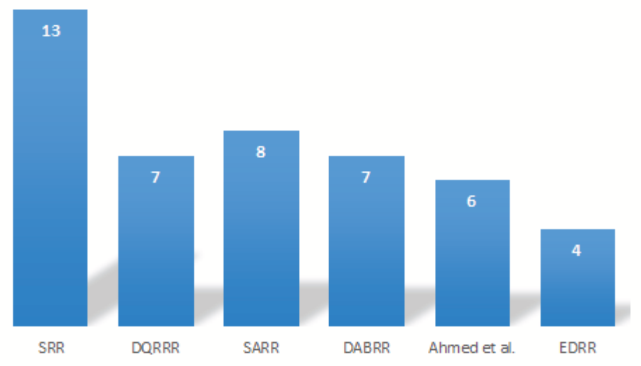

Fig. 7. Comparison of context switches

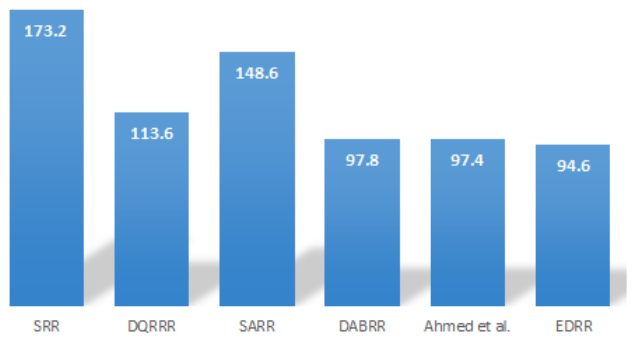

Fig. 8. Comparison of Average Waiting Time

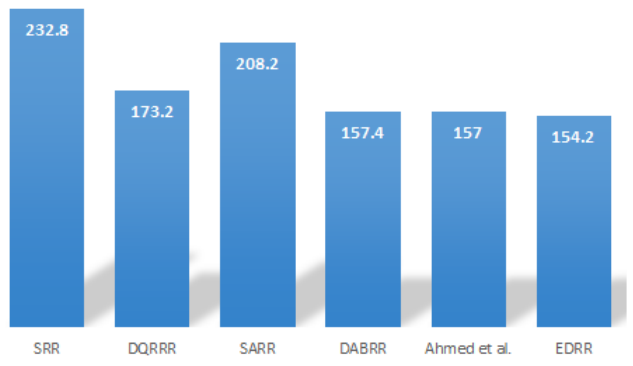

Fig. 9. Comparison of Average Turnaround Time

## VI. Conclusion

In this paper, we presented an efficient dynamic round robin algorithm that yields best effieceny performance such as context switches, average waiting and turnaround times without taking advantage of sorting the ready queue and SJF that itsef is a difficult task. Decreasing all three aforementioned constraints along with algorithm running time, the throughput is increased and pre-knoweldge is avoided. In the end, we analyzed our algorithm with some other designed algorithms to compare our superior efficiency with those sorting based algorithms.

## References

[1] Silberschatz, A., P.B. Galvin and G. Gagne, 2004. Operating Systems Concepts. 7th Edn., John Wiley and Sons, USA., ISBN: 13: 978-0471694663.

[2] A. R. Dash. S. K. Sahu and S. K. Samantra, “A Optimized Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum”, International Journl of Computer Science, Engineering and Information Technology (IJCSEIT),Vol. 5,No. 1, February 2015.

[3] Rami J. Matarneh,“Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of Now Running Processes”, American J. of Applied Sciences 6(10): 1831 -1837,2009.

[4] H. S. Behera, R. Mohanty and D. Nayek, “A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis”, International Journal of Computer Applications (0097 - 8887),Vol. 5,No. 5,pp. lOIS, August 2010.

[5] A. Noon,A. Kalakech and S. Kadry,“A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average”, International Journal of Computer Science Issues (T.TSI),Vol. 3,Issue 3,No. 1,pp. 224-229,May 2011.

[6] 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), 2015, pp. 98-104.