International Journal of Electrical and Computer Engineering (IJECE)
Vol. 9, No. 1, February 2019, pp. 190~202
ISSN: 2088-8708, DOI: 10.11591/ijece.v9i1.pp190-202

Priority based round robin (PBRR) CPU scheduling algorithm

Sonia Zouaoui[1], Lotfi Boussaid[2], Abdellatif Mtibaa[3]
[1,2,3]Laboratory of Electronics and Micro-electronics, University of Monastir, Tunisia
[2,3]National Engineering School of Monastir, University of Monastir, Tunisia
[1]National Engineering School of Sousse, Tunisia

Article Info

Article history:

Received Oct 21, 2017
Revised Jul 10, 2018
Accepted Jul 24, 2018

Keywords:

Average turnaround time
Average waiting time
Priority based
Round robin (RR)
Scheduling algorithms

ABSTRACT

This paper introduce a new approach for scheduling algorithms which aim to improve real time operating system CPU performance. This new approach of CPU Scheduling algorithm is based on the combination of round-robin (RR) and Priority based (PB) scheduling algorithms. This solution maintains the advantage of simple round robin scheduling algorithm, which is reducing starvation and integrates the advantage of priority scheduling. The proposed algorithm implements the concept of time quantum and assigning as well priority index to the processes. Existing round robin CPU scheduling algorithm cannot be dedicated to real time operating system due to their large waiting time, large response time, and large turnaround time and less throughput. This new algorithm improves all the drawbacks of round robin CPU scheduling algorithm. In addition, this paper presents analysis comparing proposed algorithm with existing round robin scheduling algorithm focusing on average waiting time and average turnaround time.

Corresponding Author:

Sonia Zouaoui,
Laboratory of Electronics and Micro-electronics,
University of Monastir, Tunisia.
Email: sonia.zouaoui87@gmail.com

1. INTRODUCTION

Scheduling is the most important service of an operating system; it gives processes access to system resources. Many requirements like fast computing, multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously) arise the need of scheduling algorithms [1]. Scheduling is in the heart of an operating system. It presents a fundamental function, which selects the process to run when there are multiple runnable processes. There are varieties of scheduling algorithms such as First Come First Served (FCFS), Shortest Job First (SJF), Round Robin (RR), Priority Based Scheduling, etc. [2, 3].

Due to their poor performance, the majority of these algorithms are rarely used in real time operating systems except for the Round Robin scheduling. In CPU scheduling, a number of assumption are taken into consideration, which are as follows [4, 5]:

a. Job pool consists of runnable processes waiting for the CPU.
b. All processes are independent and compete for resources.
c. The function of the scheduler is to fairly allocate the limited resources of CPU to the different processes and in a way that optimizes some performance criteria.

The scheduler, which constitutes the heart of the kernel, plays a fundamental role in selecting the appropriate process to be run. In this context, an operating system can be characterized according to three different types of schedulers: a long term, a mid-term or medium term and a short-term scheduler as shown in Figure 1.

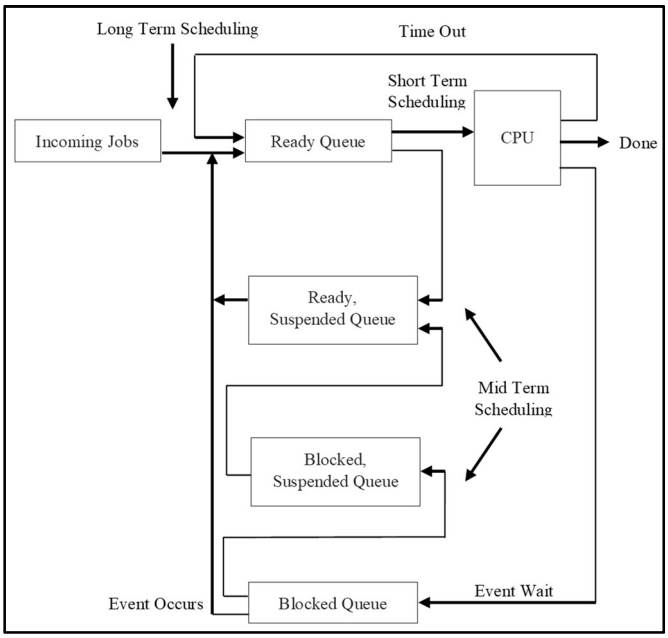

Figure 1. Various types of schedulers

For long-term scheduler, it loads processes in memory for execution after selecting them from the job pool. Concerning the short-term scheduler, it allocates the CPU to one process from those ready to be executed. For medium-term scheduler, it takes off processes from main memory and place them on secondary memory (such as a disk drive) or vice versa. This is commonly referred to as "Swapping" of processes in memory [2].

The efficiency of the Round Robin Scheduling Algorithm depends on the time quantum size. Firstly, if the quantum of time is extremely large, it decreases the response time and it behaves similar to FCFS algorithm. In the other hand, if the quantum of time is extremely small this causes many context switches, which decreases the CPU efficiency.

2. RELATED WORKS

In the recent years, a number of CPU scheduling mechanisms have been developed for predictable allocation of processor. Extracting advantages of each algorithm and try to mix them to lead to the perfect algorithm according to a specific situation.

In Mishra, an improved Round Robin scheduler is developed named Improved Round Robin (IRR) CPU scheduling algorithm. It works similar to Round Robin (RR) with a small improvement [6]. IRR picks the first process from the ready queue and allocate the CPU to it for a time interval of up to one QT. Every time a process accomplish its QT, it checks the remaining CPU burst time of the currently running process. If the remaining CPU burst time of the currently running process is less than one QT, the CPU again allocated to the currently running process for remaining CPU burst time.

To evaluate the performance, let's consider the following ready queue as shown in Table 1. For simple Round Robin algorithm, the Gantt chart is shown in Figure 2 for QT=10 ms. Concerning Improved Round Robin algorithm (IRR), the Gantt chart will be as follow as shown in Figure 3. Performances of the two algorithms are resumed in Table 2 regarding to Average WT and Average TT.

Table 1. Process's Id and Burst Time

Process ID	Burst Time (ms)
P1	5
P2	12
P3	23
P4	26
P5	34

Table 2. Comparison of RR and IRR

Algorithm	Average WT (ms)	Average TT (ms)
RR	38.4	57.8
IRR	30.4	49.8

	P1	P2	P3	P4	P5	P2	P3	P4	P5	P4	P5	P5
0	5	15	25	35	45	47	57	67	77	83	93	97

Figure 2. Gantt chart for simple RR

	P1	P2	P3	P4	P5	P2	P3	P4	P5	P4	P5	P5
0	5	15	25	35	45	47	57	67	77	83	93	97

Figure 3. Gantt chart for IRR

The proposed IRR in CPU scheduling algorithm is giving better performances than RR. After improvement in RR, it has been found that the WT and TT have been reduced drastically [6]. A modified round robin algorithm, proposed by [7], consists of a mixture between shortest job first and round robin algorithms. In this new algorithm, the QT takes the burst time of mid process when the number of processes is odd else, it takes the average time of burst time of all processes.

To evaluate the performance, let's consider the following ready queue as shown in Table 3. For simple Round Robin algorithm, the Gantt chart is shown in Figure 4 with QT=25 ms. Concerning Modified Round Robin algorithm (MRR), the Gantt chart will be as follow as shown in Figure 5. Performances of the two algorithms are resumed in Table 4. regarding to average WT and average TT.

Table 3. Process's Id and Burst Time

Process ID	Burst Time (ms)
P1	14
P2	45
P3	36
P4	25
P5	77

Table 4. Comparison of RR and MRR

Algorithm	Average WT (ms)	Average TT (ms)
RR	70.2	109.6
MRR	56.8	96.2

	P1	P2	P3	P4	P5	P2	P3	P5	P5
0	14	39	64	89	114	134	145	170	195

Figure 4. Gantt chart for simple RR

	P1	P2	P3	P4	P5	P2	P5	P5
0	14	50	86	111	147	156	192	197

Figure 5. Gantt chart for MRR

It is concluded from the above experiments that the proposed algorithm MRR performs better than simple RR in terms of performance metrics such as average WT and average TT. In the work of [8], the proposed algorithm is based on the integration of Round Robin and priority scheduling algorithm and also implements the concept of aging by attributing new process priorities. First, the CPU is allocated to every process in round robin fashion with a given priority and quantum. Then, processes are sorted in increasing order according to their remaining CPU burst time in the ready queue. Consequently, new priorities are assigned; the process with shortest remaining CPU burst is assigned with highest priority. Each process gets the control of the CPU until they finished their execution. This new approach improves the performance of CPU in real time operating system.

To evaluate the performance, let's consider the following ready queue as shown in Table 5. For simple Round Robin algorithm with QT=5ms, we obtain the following Gantt chart as shown in Figure 6. Concerning Priority based Round Robin algorithm (PRR), tasks are executed according to their priority, and the Gantt chart will be as follow as shown in Figure 7. Performances of the two algorithms are resumed in Table 6. regarding to average WT and Average TT.

Table 5. Process's Id and Burst Time

Process ID	Burst Time (ms)	Priority
P1	22	4
P2	18	2
P3	9	1
P4	10	3
P5	4	5

Table 6. Comparison of RR and PRR

Algorithm	Average WT (ms)	Average TT (ms)
RR	33.2	45.8
PRR	26.2	38.8

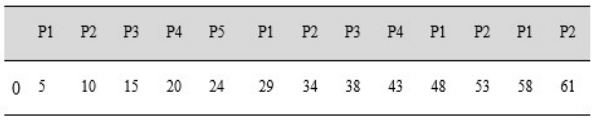

Figure 6. Gantt chart for simple RR

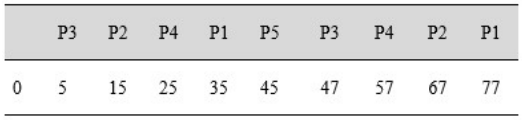

Figure 7. Gantt chart for PRR

The proposed algorithm improves all the drawbacks of round robin CPU scheduling algorithm. It retains the advantage of round robin in reducing starvation and also integrates the advantage of priority scheduling. In Abdulrahim *et al.* a New Improved Round Robin (NIRR) was developed [9]. This algorithm is an improved version of Improved Round Robin (IRR) algorithm mentioned above in [6]. This algorithm holds processes according to their arrival times in a queue named ARRIVE queue. In the other hand, other processes are in a queue called REQUEST queue waiting their turn to occupy the CPU. The time quantum is considered as the average of burst times of the processes in the REQUEST queue.

To evaluate the performance, let's consider the following ready queue as shown in Table 7. For simple Round Robin algorithm with QT=50ms, we obtain the following Gantt chart as shown in Figure 8. Concerning New Improved Round Robin algorithm (NIRR), the Gantt chart is as follows as shown in Figure 9. Performances of the two algorithms are resumed in Table 8 regarding to the average WT and the average TT.

Table 7. Process's with Its Id and Burst Time

Process ID	Burst Time (ms)
P1	23
P2	75
P3	93
P4	48
P5	2

Table 8. Comparison of RR and NIRR

Algorithm	Average WT (ms)	Average TT (ms)
RR	113	161.2
NIRR	53.8	102

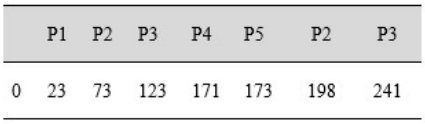

Figure 8. Gantt chart for simple RR

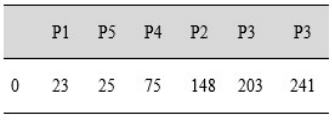

Figure 9. Gantt chart for NIRR

In Emilio an algorithm called “Modulo Based Round Robin Algorithm” is proposed. This algorithm is based on Round Robin fashion an intelligent time quantum and then assigns priority to the processes [10]. This algorithm start by calculating the average of CPU burst of all the processes (P) and then compute for each process the burst time modulo P which is named (M). After that processes are sorted according to the value of M and then the time quantum (QT) is considered equal to P.

To evaluate the performance, let's consider the following ready queue as shown in Table 9 (All processes arrive at t=0 and QT=10 ms). For simple Round Robin algorithm, we obtain the following Gantt chart as shown in Figure 10. For “Modulo Based Round Robin Algorithm”, the Gantt chart is as follows as shown in Figure 11. Performances of the two algorithms are resumed in Table 10 regarding to the average WT and the average TT.

Table 9. Process's Id and Burst Time

Process ID	Burst Time (ms)
P1	10
P2	20
P3	30
P4	40
P5	50

Table 10. Comparison of RR and “Modulo Based Round Robin Algorithm”

Algorithm	Average WT (ms)	Average TT (ms)
RR	60	90
NIRR	52	82

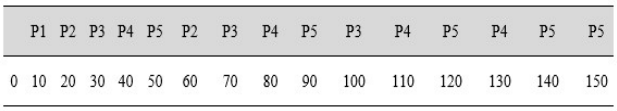

Figure 10. Gantt chart for simple RR

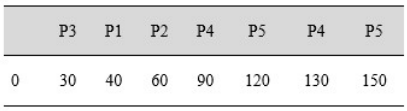

Figure 11. Gantt chart for "Modulo Based Round Robin Algorithm"

3. REAL TIME OPERATING SYSTEM (RTOS)

Real Time Operating System (RTOS) is a reactive OS that must respond continuously to stimuli from a process that seeks to control. A real-time system is a reactive system that must meet time constraints [11]. A real-time system must be able to process information from the process within a period that does not affect the process control. React too late can lead to catastrophic consequences for the system itself or the process. Compliance with time constraints is the main constraint to satisfy. The validity of a real time system depends not only on the results of the treatment carried out but also the temporal aspect (a fair calculation but out of time is an invalid calculation).

3.1. Real-time systems classification

The critical time constraints led to classify the real-time systems in the following three categories [12]:

a. Real time strict system: a system that is subject to strict time constraints, that is to say for which the slightest mistake time can have devastating human and economic consequences. Most applications in the avionics field, automobile, etc., are strict real time;
b. Real-time flexible system: a system that is subject to flexible time constraints, a number of timing errors can be tolerated.
c. Real-time mixed system: a system that is subject to strict and flexible time constraints.

3.2. Real-time task

A real-time task consists of a set of instructions that can be run in sequence on one or more processors and meet time constraints. In the following, we will assume that a task will not run in parallel. It can be repeated any number of times, possibly infinite. Each of these performances is called instance or work ("job"). A real-time system consists of a set of real-time tasks subject to real-time constraints (Real time constraints).

A real-time task can be:

a. Periodic: its instances (versions) are repeated indefinitely and there is a constant time between two successive activations of instances referred period.
b. Sporadic: its instances (versions) are repeated indefinitely and there is a minimum time between two successive instances.
c. Aperiodic: there is no correlation between successive instances.

3.3 Periodic tasks

The classic model of periodic tasks called Liu and Layland models, the most used in modeling real-time systems [13]. This model allows defining multiple settings for a job. These parameters are of two types: static parameters for the task itself and the dynamic parameters on each instance of the task as shown in Figure 12. The basic static parameters of a periodic task $\tau_i$is:

$$\tau_i = (R_i, C_i, D_i, T_i) \quad (1)$$

R_i: (release time) date of first activation of task i, when τ_i can start its first performance.
C_i: (computing time): execution time of τ_i. This parameter is considered in several works on real-time scheduling as the worst case execution time (WCET for worst-case execution time) on the processor on which it will run.
D_i: relative maturity or critical time frame for each activation of τ_i.
T_i: implementation period τ_i.

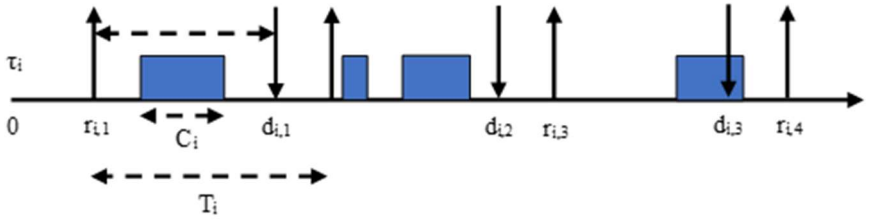

Figure 12. Classic model in periodic tasks

Other static parameters are derived from the basic ones:

$U_i = \frac{C_i}{T_i}$, CPU utilization factor for τ_i, $U_i \leq 1$.

$CH_i = \frac{C_i}{D_i}$, the density of the task τ_i, $CH_i \leq 1$.

Concrete/ no concrete tasks

If all the first activation dates of all jobs are known, it is said that the tasks are concrete. On the contrary, if we do not know the dates of first activation, it is said that these tasks are not concrete.

a. Synchronous/ asynchronous tasks
   If all tasks are practical and have the same date of first activation, it is said that the tasks are synchronized activations. Otherwise, they are asynchronous
b. Real-time constraints
   In real-time scheduling, tasks can be subject to several constraints such as the constraints of deadlines, strict periodicity, dependencies and precedence, etc.
c. Deadlines
   The constraints of deadlines allow expressing a condition of the end date of implementation at latest of a given task. Consider a system of periodic tasks, according to the relationship between the period T_i and the deadline for each task i D_i, we distinguish three types of deadlines:
   - $D_i = T_i$: each activation of the task τ_i must be executed before the next activation. We talk about deadlines on tasks activations, implicit deadlines or deadlines on requests.
   - $D_i \leq T_i$: each activation of the task τ_i must be executed on or before a date less than or equal to the date of its next activation. We talk about stress at work deadlines.
   - $D_i \neq T_i$: there is no correlation between at latest end date of execution of τ_i upon activation and its next activation of τ_i. One can have ($D_i \leq T_i$) or ($D_i > T_i$), we speak here about arbitrary tasks deadlines.
d. Strict periodicity
   Consider a periodic task τ_i in a real time task system, strict periodicity constraint requires that the time elapsed between two consecutive beginnings of execution dates s_i^k and s_i^{k+1} corresponds exactly to the period of the task τ_i. The advantage of this constraint is that knowledge of the actual start date of the first instance s_i^1 implies knowledge of the effectivestart date of all subsequent instances of the same task [14], this is expressed by the relationship:

   $$s_i^{k+1} = s_i^1 + kT_i \ (k \geq 1) \quad (2)$$

e. Dependencies between tasks
   A dependency between two tasks i and j can be of two types: A precedence dependency and/or data dependency. A precedence dependency between (i, j) requires that the task j started running after the task i have completely finished running [14, 15, 16, 17]. The precedence of constraints is indirectly real-time constraints and we said that the task i is a predecessor of the task j and j is a successor of i. If task i runs exactly once before a run of task j, we have then a simple precedence constraint if not it's a wide precedence [17, 2, 18]. A data dependency means the task i produces a result that is consumed by j [14, 15], this dependence inevitably leads to precedence between tasks. Tasks are called independent if they are defined only by their temporal parameters.
f. Latency
   Latency is defined for dependent tasks by transitivity in the case of a path tasks. Let's suppose τ_i and τ_j two tasks, the latency between i and j denoted L (τ_i, τ_j) is the time between the start of execution of τ_i and the end of execution of τ_j.

4. SCHEDULING

4.1. Scheduling objectives

a. When designing a scheduling algorithm, a system designer must take into consideration many factors such as the kind of systems used and user's needs.
b. Maximize throughput: Maximizing the throughput of a scheduler is made by servicing the maximum number of processes per unit of time.
c. Avoid an infinite blocking state or starvation: Avoiding an infinite blocking state or starvation is avoiding process to stuck in waiting state for unbounded time before or while process service.
d. Minimize overhead: Using system resources in an effective manner to reduce system overhead (system overhead cause resources wastage). So overall system performance improves greatly.
e. Enforcement of priorities: If a system is based on processes priorities, the scheduler shall privilege higher priority processes.
f. Achieve balance between response and utilization: System resources shall be kept busy by the scheduler.

The scheduler can prevent starvation through the concept of aging and be able to increase throughput by promoting processes having a short burst time and can be satisfied quickly. Also, processes whose completion because other processes to run shall be favored by the scheduler to accomplish goals previously sited.

4.2. Scheduling parameters

Each CPU scheduling algorithm has its own proprieties, and choosing a particular algorithm may favor one class of process over the other. Many criteria must be considered for comparing CPU scheduling algorithms performance. Those criteria include the following:

a. CPU utilization: the maximum rate of keeping the CPU busy doing useful work.
b. Throughput: The number of jobs processed per time unit.
c. Turnaround time: The time needed for the execution of one process.
d. Waiting time: The time spent by the process waiting in ready queue.
e. Response time: The time between submission of the request and the first response.
f. The sharing of the processor and resources introduced several states for a task as shown in Figure 13:
    1. New: the task is not yet activated.
    2. Ready: the task is enabled and has all the resources it needs to run.
    3. Waiting: the task is waiting for resources.
    4. Running: the task runs.
    5. Terminated: the task has no current application, it has terminated.

The scheduler is responsible for the transition from one task state to another. For each invocation, the scheduler updates the list of ready tasks by including all active tasks and who have their resources and removing tasks that ended their performances or blocked by waiting a resource. Then among the ready tasks, the scheduler selects the highest priority task to run. Thus, a task in the new state can move to the ready state. A task in the ready state may go to running state or waiting state. A task in running status may return to the ready state if it is preempted by another higher-priority task, it can go to the waiting state if it is waiting for resource liberation or has finished executing, it passes in the passive state.

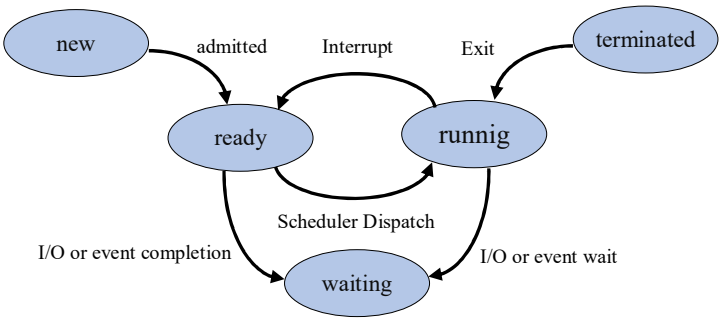

Figure 13. Process state transition diagram

A task can move from the waiting state to the ready state, and finally a task can move from the passive state to the ready state. Figure 13 provides an illustration of the different states and their transitions [2].

5. EXISTING CPU SCHEDULING ALGORITHMS

There are varieties of CPU Scheduling algorithms. The most and commonly used are explained.

a. First-come first-served (FIFO) scheduling policy
It is a policy of seniority without requisition: The CPU is allocated according to the process submission order. In this policy, processes of low execution time can be penalized because a long process precedes them in the queue.

b. Scheduling policy "shorter first" scheduling policy
This policy tries to remedy the disadvantage mentioned for the previous policy. The CPU is allocated to the process of smaller execution time. This policy is also a policy without requisition. It has the property of minimizing the average response time for all scheduling algorithms without requisitioning. It penalizes long work. It also makes it necessary to estimate the duration of the processes, which are not usually known. There is a version with requisition of this policy called "time remaining first shortest": in this case, the executing process restores the processor when a new execution time process less than its remaining execution time becomes ready [18].

c. Round robin scheduling policy
Each process present in the queue of ready processes acquires the processor in turn for a maximum of a time equal to the quantum of time. If the process has completed its execution before the end of the quantum, it releases the processor and the next process in the queue of the ready processes is elected. If the process has not completed before the end of the quantum, it loses the processor and is reinserted at the end of the process. This turnstile policy is usually used in timeshare systems. Its performance largely depends on the size of the quantum. Too large quantum increases the response times while a too small quantum multiplies the context switches until they are not negligible [19].

d. Priority-based Scheduling:
A fixed priority rank is assigned to each process and the scheduler sort processes basing on their priorities. Process with highest priority is putted on the head of the queue and the process with the lowest priority in the tail [20].

e. Rate Monotonic (RM)
RM scheduling algorithm was introduced by Liu and Layland in 1973 [13]. This is a preemptive scheduling algorithm which applies to independent periodic tasks, and due upon request (Ti=Di). The task priority is inversely proportional to its period, that is to say, the longer the period of a task is, the higher is the priority. This algorithm is optimal in the class of algorithms for fixed priority preemptive independent tasks due on synchronous request. A sufficient condition for schedulability of the RM algorithm for a set of periodic tasks Γn maturing on request is given by [13]:

$$\sum_{i=1}^{n} \frac{Ci}{Ti} \leq n(2^{\frac{1}{n}} - 1) \quad (3)$$

f. Deadline Monotonic (DM)
The DM scheduling algorithm was introduced by Leung and Whitehead in 1982 for conditioned deadline tasks [21]. The task priority is inversely proportional to its relative deadline, that is to say, the longer the relative deadline is thehigher isthe priority. This algorithm is optimal in the class of preemptive and fixed priority algorithms for independent preemptive and conditioned deadline tasks (Di ≤ Ti). The sufficient condition for schedulability of the DM algorithm for a set of periodic tasks Γn due to stress is given by:

$$\forall \tau_i \in \Gamma_n,\ C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{D_i}{T_j} \right\rceil . C_i \leq D_i \quad (4)$$

With hp(τi) is the set of Γn tasks of priorities higher than or equal to Γn, not including Γn.
Dynamic priorities. A dynamic priority changes during the execution of an instance [22].

a. Earliest Deadline First (EDF)
The EDF scheduling algorithm was introduced by Lui and Layland in 1973 [13]. It is a scheduling algorithm which can be preemptive or non-preemptive and applied for independent periodic tasks (Ti = Di). The highest priority at time t is allocated to the task with the closest absolute deadline.

EDF is optimal for independent and preemptive tasks. A necessary and sufficient condition of schedulability of EDF for a set of periodic tasks Γn is given by:

$$\forall \tau_i \in \Gamma_n \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \quad (5)$$

b. Least-Laxity First (LLF)
The LLF scheduling algorithm is based on the laxity. The task whose laxity is the lowest compared to all the ready tasks have the highest priority. This algorithm is optimal for independent and preemptive tasks.
The condition of schedulability of LLF and the EDF are the same. That is to say that the necessary and sufficient condition of schedulability of LLF for a set of periodic tasks Γnis given by:

$$\forall \tau_i \in \Gamma_n \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \quad (6)$$

A dynamic priority changes during the execution of an instance. The most used scheduling algorithm for dynamic priorities is "Least laxity Fisrt".

6. DRAWBACKS OF ROUND ROBIN SCHEDULING ALGORITHMS

Round Robin scheduling algorithm has many disadvantages, which are as following:

6.1. High Average Waiting Time

For round robin architecture, the process spends the time in the ready queue waiting his turn to own the processor. Due to the presence of time quantum, processes are pushed to leave the processor and return to the waiting state. This procedure produces a high average waiting time, which presents the main disadvantage.

6.2. Low throughput

Throughput present the number of process completed per time unit. Due to its time slice, Round Robin is characterized by a high number of context switches, which leads to overall degradation of the system performance (throughput).

6.3. Context switch

Ones the time slice finished, the process is forced to leave the CPU. The scheduler stores the context of the current process in stack or a register and allots the CPU to the next process in the ready queue. This concept is known by context switching which leads to time wastage and scheduler overhead.

6.4. High response time

Response time is defined as the time between submission of a request and the first CPU response. In general, round robin made larger response time, which causes system performance degradation. To achieve high performance, the reduction of response time shall be indispensable.

6.5. Very high turnaround time

Turnaround time is the time between submission of a process and its completion. Round Robin scheduling algorithm is characterized by a high turnaround time. So as a result, to improve the system performance this parameter should be reduced.

7. EFFICIENCY OF EXISTING APPROACHES

Scheduling processes in a fair manner and producing results in minimum average waiting time and turnaround time are the main criteria of an efficient algorithm. In different research papers, the issue of time quantum and priority decision is the bottleneck in various CPU scheduling algorithm. Finding the optimal time quantum in round robin algorithm is a difficult problem to solve. The real meaning of dynamicity in CPU scheduling algorithms is not goal to attempt. A variety of objectives are to be taken into consideration and solutions must satisfy most of the performance criteria. Solutions must be universally accepted by most of the operating system. Concerning dynamic time quantum and dynamic priority calculation, many researchers are carried out and some of them paid attention towards the combination of two. Other

researchers pay big attention to response time and some of them even used artificial intelligence in determining dynamic time quantum or priority. In general, measurement are not far from outliers. So giving the optimum results cannot be obvious for the set of processes having CPU burst time. The presence of outliers in the list is obvious especially when the CPU always tries to keep a mix of CPU bound and I/O bound processes. Decreasing the outlier makes the data more exact when finding mean or median value for determining time quantum.

8. PROPOSED ALGORITHM

This approach of Round Robin Scheduling Algorithm emphasizes on simple Round Robin Algorithm drawbacks. This kind of scheduling gives the same amount of time to all processes. Processes are executed in first come first served way. Round Robin is not efficient with processes with small execution time. Therefore, as a result, the waiting time and response time of processes increase and consequently the system throughput decreases. In this proposed algorithm, we have implemented RR algorithm while taking into account priority based for tasks management. In fact, a priority index is assigned to each Process. Processes in ready queue are sorted according to their priorities. Process with small priority index are placed in the head of the ready queue and so on. The proposed algorithm solves the problem of higher average waiting time, turnaround time, response time and more context switches and thereby improves the system performance. The proposed algorithm is described in the following flowchart as shown in Figure 14.

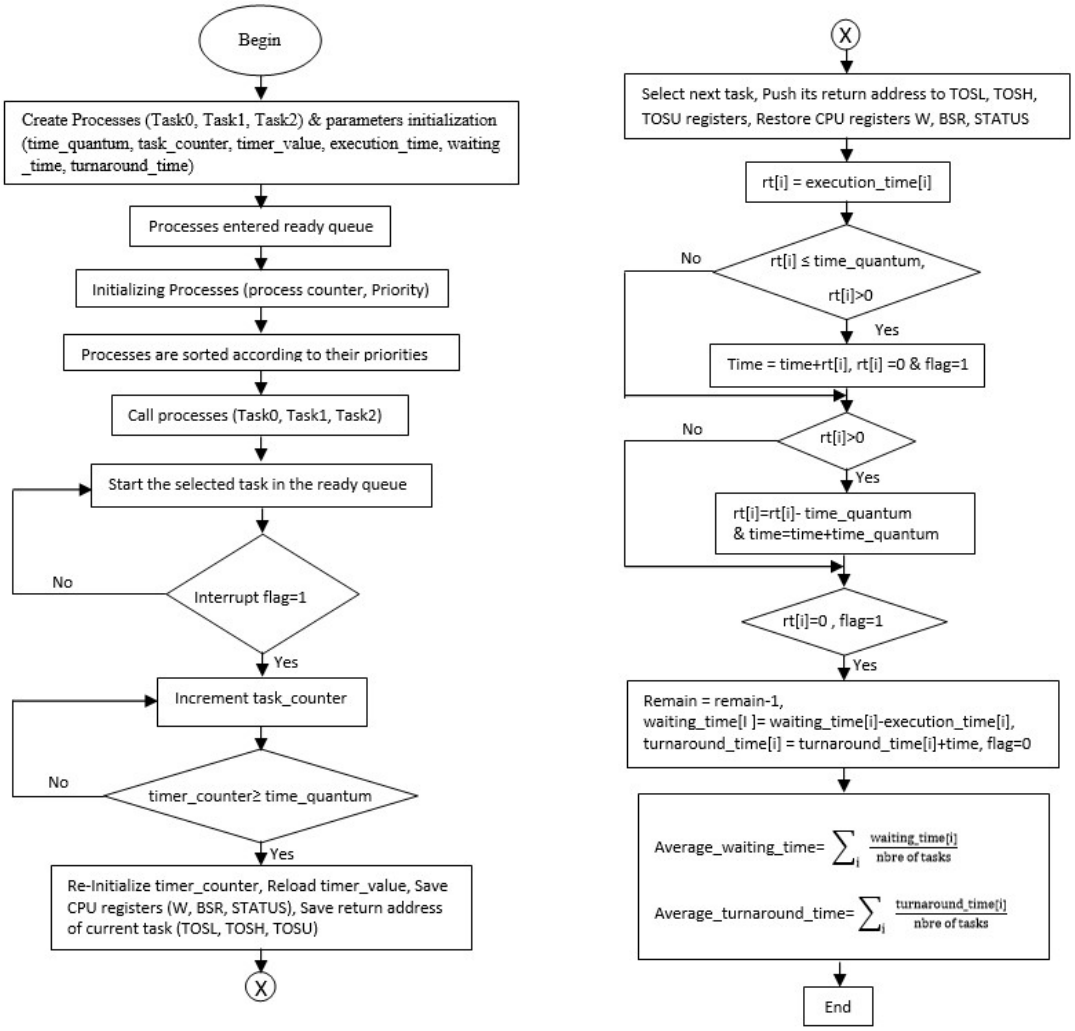

Figure 14. PBRR flowchart

8.1. PBRR illustration by taking example

We have three processes P0, P1and P2 in ready queue arriving at time 0 with burst time 10, 3 and 2 respectively. Each process has its own priority index 2, 1 and 3respectively (the lowest index is associated to the highest priority).We consider Time quantum (TQ) is assumed4 milliseconds (ms). In step1, the scheduler selects the process, which has the highest priority and put it in head of the ready queue and so on. In the end, the ready queue is sorted according to priority index of each process.

The process P1 has the highest priority, so the scheduler assigns P1 to CPU. After execution of allocated process for time interval of 3ms, P1 has finished execution, it will be removed from the ready queue. Next process in the sorted ready queue, which has the highest priority, is P0 with 10ms CPU burst time. CPU will be allocated to P0 for a time interval of 4ms.P0 has a remaining CPU burst time to 6ms. It will be compared to time quantum so it is less than the specified TQ value soP1is put at the tail of the ready queue to be allocated again to CPU for remaining time interval of 6ms. Next process in the ready queue, which has highest priority is P2 with 2ms CPU burst time. The scheduler will allocate P2 to the CPU for a time interval of 2ms. Since the CPU burst time of P2 is less than the TQ, P2 has finished execution, it will be removed from the ready queue. Now, P0 is the only process that didn't finish its execution. So P0 will be allocated to the CPU again for 4ms. P0 remaining burst time became 2ms. Since there are no other processes in the ready queue, P0 is reallocated to the CPU for the remaining CPU burst time. P0 finish its executions and will be removed from the ready queue.

When using PBRR scheduling algorithm, the waiting time is 5 ms for P0, 0ms for P1and 7 ms for P2, as result the average waiting time is 4ms. It should be noted that the average waiting time is 5.33 ms when using simple RR scheduling algorithm. This average waiting time was significantly improved when using PBRR scheduling algorithm for the same set of processes and features.

8.2. Experimental Result

The environment in which the execution takes place is a single processor environment and all the processes are independent. All the processes have the same arrival time. All the attributes like burst time, number of processes, priority and the time slice of all the processes are known before submitting the processes to the processor. The time quantum is taken in milliseconds. For experimental observation it is considered that the arrival time of all processes is zero.

8.2.1. Case Studies:

a. Example 1

Suppose RR time quantum =4, Table 11 shows processes with their burst times and priorities. According to simple Round-Robin Algorithm the Gantt chart obtained as in Figure 15.

Table 11. Processes with Its Id, Burst Time and priority

Process ID	Burst Time (ms)	Priority
P0	10	2
P1	3	1
P2	2	3

	P0	P1	P2	P0	P0
0	4	7	9	13	15

Figure 15. Gantt chart for simple RR

Total average waiting time=5.33
Average turnaround time=10.33
According to simple PBRR Algorithm the Gantt chart obtained as in Figure 16.

	P1	P0	P2	P0	P0
0	3	7	9	13	15

Figure 16. Gantt chart for PBRR algorithm

Then we will obtain:
Total average waiting time=4
Average turnaround time=7.33
Results are resumed in Table 12 mentioned,

Table 12. Comparison of simple RR and PBRR

Algorithm	Average waiting time	Average Turnaround time	Throughput
Simple RR	5.33	10.33	Low
PBRR	4	7.33	High

b. Example 2

Suppose RR time quantum =4ms, Table 13 shows another combination of processes with their burst times and priorities. According to simple Round-Robin Algorithm the Gantt chart will be as in Figure 17.

Table 13. Processes with Its Id, Burst Time and priority

Process ID	Burst Time (ms)	Priority
P0	12	2
P1	25	3
P2	7	1

P0	P1	P2	P0	P1	P2	P0	P1	P1	P1	P1	P1
0	4	8	12	16	20	23	27	31	35	39	43	44

Figure 17. Gantt chart for simple RR

Total average waiting time=16.66
Average turnaround time=31.33
And According to simple PBRR Algorithm Algorithm the Gantt chart will be as in Figure 18.

P2	P0	P1	P2	P0	P1	P0	P1	P1	P1	P1	P1
0	4	8	12	15	19	23	27	31	35	39	43	44

Figure 18. Gantt chart for PBRR algorithm

Then we will obtain:
Total average waiting time=14
Average turnaround time=29
Results are resumed in Table 14 mentioned,

Table 14. Processes with Its Id, Burst Time and priority

Algorithm	Average waiting time	Average Turnaround time	Throughput
Simple RR	16.66	31.33	Low
PBRR	14	29	High

9. CONCLUSION

In this paper, we introduced a real time operating system and real time tasks. In addition, we present the scheduling objectives and parameters that must be considered for comparing CPU scheduling algorithms

performance. We emphasize on studying RR drawbacks like high average turnaround, high context switching, high response time, high turnaround time and low throughput. After analyzing RR performances and drawbacks, we proposed a new algorithm named Priority Based Round Robin (PBRR), which deal with drawbacks of implementing a simple round robin algorithm. Priority index is assigned to each Process. Processes in ready queue are sorted according to their priorities. This new approach is performing better than a simple RR in terms of average waiting time, average turnaround time.

For future work, we look further ahead to use hardware architecture based on FPGA. The hardware accelerators will be used during the tasks scheduling mainly in sorting processes in the ready queue. In fact, the use of a hybrid scheduler will improve system latency and determinism.

REFERENCES

[1]. William Stallings, "Operating Systems Internal and Design Principles", 5thEdition, ISBN-10: 0-13-230998, 2006.
[2]. Silberschatz, A., Peterson, J. L., and Galvin, B., "Operating System Concepts", Addison Wesley, 7th Edition, ISBN-10: 0471694665, 2006.
[3]. E.O. Oyetunji, A. E. Oluleye," Performance Assessment of Some CPU Scheduling Algorithms", *Research Journal of Information Technology*, 1(1), pp. 22-26, 2009.
[4]. John Wiley, "The Architecture of Computer Hardware and Systems Software: An Information Technology Approach*", Fourth Edition*, ISBN: 978-0471-71542-9, 2009.
[5]. Yavatkar, R. and K .Lakshman, "A CPU Scheduling Algorithm for Continuous Media Applications", *In Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video,* pp. 210-213, 1995.
[6]. Mishra, M. K. "An Improved Round Robin CPU Scheduling Algorithm." *International Journal of Global Research in Computer Science (UGC Approved Journal)*, 3(6), pp. 64-69, 2012.
[7]. Chhugani, B., & Silvester, M., "Improving Round Robin Process Scheduling Algorithm." *International Journal of Computer Applications*, 166(6), 2017.
[8]. Rajput, I. S., & Gupta, D., "A Priority Based Round Robin CPU Scheduling Algorithm for Real Time Systems," *International Journal of Innovations in Engineering and Technology*, 1(3), pp. 1-11, 2012.
[9]. Abdulrahim, A., Abdullahi, S. E., & Sahalu, J. B., "A New Improved Round Robin (NIRR) CPU Scheduling Algorithm," *International Journal of Computer Applications*, 90(4), 2014.
[10]. Emilio, M. D. P, Embedded systems design for high-speed data acquisition and control, *Springer*, 2016.
[11]. Patra, P. K., & Pradhan, P. L. "Dynamic Value Engineering Method Optimizing the Risk on Real Time Operating System." *Indonesian Journal of Electrical Engineering and Informatics (IJEEI),* 2 (2), 101-110, 2014.
[12]. Awadalla, M. H. A., "Heuristic Approach for Scheduling Dependent Real-Time Tasks." *Bulletin of Electrical Engineering and Informatics*, 4 (3), 217-230, 2015.
[13]. C. L. Liu and J W. Layland, "Scheduling Algorithms for Multiprogramming in A Hardrealtimeenvironment", ACM 20, vol. 20, pp. 46-61, 1973.
[14]. Pinedo, M. L., "Scheduling: Theory, Algorithms, and Systems," *Springer,* 2016.
[15]. J-J. Hwang, Y-C. Chow, F. D. Anger, and C-Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times", *SIAM Journal on Computing*, vol. 18, pp. 244–257, April 1989.
[16]. W. W. Chu and L. M-T. Lan, "Task Allocation and Precedence Relation for Distributed Real-Time Systems", *IEEE Transactions on Computers*, vol.C-36(18), June 1987.
[17]. J. Xu, "Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations", *IEEE Trans. Softw. Eng*., vol. 19, pp.139–154, February 1993.
[18]. Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C., "Operating Systems: Three Easy Pieces", *Chapter Scheduling Introduction, Arpaci-DusseauBooks,* ISBN, 1105979121, 2014.
[19]. A. Noon, A. Kalakechdan S. Kadry, "A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average," *IJCSI International Journal of Computer Science Issues*, vol. 8, no. 3, pp. 224-229, 2011.
[20]. I. S. Rajput,Dan D. Gupta, "A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems," *International Journal of Innovations in Engineering and Technology*, vol. 1, no. 3, pp. 1-11, 2012.
[21]. J. Y-T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks", *Performance Evaluation Journal*, vol. 2, no. 4, pp. 237-250, 1982.
[22]. P. Singh, V. Singh dan A. Pandey, "Analysis and Comparison of CPU Scheduling Algorithms," *International Journal of Emerging Technology and Advanced Engineering*, vol. 4, no. 1, pp. 91-95, 2014.