Volume 4, Issue 7, July 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Comparison of Different Task Scheduling Algorithms in RTOS: A Survey

Dr. D. G. Harkut
Associate Prof. PRMCEAM,
Bandera, India

Prof. Anuj M. Agrawal
M.E. 2nd yr, PRMCAM,
Bandera, India

Abstract: ***In this paper, we survey some of the classic real-time scheduling algorithms. Many papers have been published in the field of real-time scheduling. The problem of scheduling is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed service. The quality of real-time scheduling algorithm has a direct impact on real-time system's working. We studied popular scheduling algorithms mainly EDF and RM for periodic tasks with hard deadlines. After that, we describe performance parameters we use to compare the performances of the various algorithms. We observe that the choice of a scheduling algorithm is important in designing a real-time system. We conclude by discussing the results of the survey and suggest future research directions in the field of RTOS.***

Keywords: CPU, RTOS, LLF, EDF, PCP

I. INTRODUCTION

The most critical constraint of real-time systems is that the correctness of such systems depends not only on the computed results but also on the time at which results are produced. In other words, real-time systems have timing requirements that must be guaranteed. This leads to the notion of deadline which is a common thread among all real-time system models and the core of the difference between real-time systems and time-sharing systems. The deadline of a task is the point in time before which the task must complete its execution. There can be three types of deadlines, which are mentioned below. **Soft Deadline:** If the results produced after the deadline has passed and are still useful then this type of deadline is known as soft deadline. Reservation systems come under this category. **Firm deadline:** This deadline is one in which the results produced after the deadline is missed is of no utility. Infrequent deadline misses are tolerable. These types of deadlines are used in systems which are performing some important operations. **Hard deadline:** If catastrophe results on missing the deadline then this type of deadline is known as hard deadline. The systems which are performing critical applications like air traffic control come under this category. There are two kinds of real-time tasks, depending on their arrival pattern: periodic tasks (the task has a regular inter-arrival time called the period, a deadline and a computation time) and aperiodic tasks (the task can arrive at any time; such a task is characterized by a computation time and a deadline; the latter is usually considered as soft). An essential component of a computer system is the scheduling mechanism that is the strategy by which the system decides which task should be executed at any given time. The problem of real-time scheduling is different from that of multiprogramming time-sharing scheduling because of the role of timing constraints in the evaluation of the system performance. Normal multiprogramming time-sharing systems are expected to process multiple job streams simultaneously, so the scheduling of these jobs has the goals of maximizing throughput and maintaining fairness. In real-time systems the primary performance is not to maximize throughput or maintain fairness, but instead to perform critical operations within a set of user-defined critical time constraints.

.

II. RELATED WORK

This paper makes a really good point of comparing different scheduling algorithms for RTOS under different parameters. Liu and Layland [2] came up with Rate Monotonic scheduling (priority driven scheduling) of real-time operating systems which is premier scheduling algorithm implemented in almost all the RTOSes. Better scheduling algorithm - Earliest Deadline First (deadline driven scheduling) is too complex to be implemented in real-time operating system [18].Timmerman [10] describes the framework for evaluation of realtime operating systems. This article makes a really good point of comparing RTOS under different load conditions. Baskiyar [15] and et al. have made an extensive survey on memory management and scheduling in RTOS. A worst case response time analysis of real time tasks under hierarchical fixed priority pre-emptive scheduling is done by Bril and Cuijpers. Yaasuwanth [3] and et. al. have developed an modified RR algorithm for scheduling in real time systems. Recently, a number of CPU scheduling algorithms have been developed for predictable allocation of processor [12]. From the work done by the various researchers in the field of real time scheduling; so far, it has been observed that

a. Scheduling should be done in order to guarantee the schedule of the processes fairly and throughput must be maximum. **b.** Real time scheduling algorithms are always pre-emptive which can perform better if the pre-emption is limited. **c.** Static priority scheduling algorithms are used for scheduling real time tasks for maximum CPU utilization but

it can be increased more using dynamic priorities. **d.** The schedulability of scheduling algorithm must be checked using schedulability tests. **e.** Starvation should not be there which means a particular process should not be held indefinitely. Allocation of resources should be such that all the processes get proper CPU time in order to prevent starvation. **f.** In case of priority based algorithms, there should be fairness in the pre-emption policy. Low priority tasks should not wait indefinitely because of higher priority tasks.

III. CLASSIFICATION OF SCHEDULING ALGORITHMS

A *scheduler* provides a policy for ordering the execution of tasks on the processor, according to some criteria. Schedulers produce a schedule for a given set of processes. There are several classifications of schedulers. Here are the most important:

- **Optimal or non-optimal:** An optimal scheduler can schedule a task set if the task set is schedulable by some scheduler.

- **Preemptive or non-preemptive:** A preemptive scheduler can decide to suspend a task (before finishing its execution) and restart it later, generally, because a higher priority task becomes ready. Non-preemptive schedulers do not suspend tasks in this way. Once a task has started, it cannot be suspended involuntarily.

- **Static or dynamic:** Static schedulers calculate the execution order of tasks before run-time. It requires knowledge of task characteristics but produces little run-time overhead. However, it cannot deal with aperiodic or non-predicted events. Some references about this kind of schedulers can be found in [20]. Dynamic schedulers, on the contrary, make decisions during the run-time of the system. This allows to design a more flexible system, but it means some overhead.

Priority-driven schedulers are dynamic schedulers which decide what task to execute depending on the importance of the task, called *priority*. This kind of schedulers can be also fixed or dynamic, depending on whether the task priority can vary during run-time. Because Priority-driven schedulers have been widely used in real-time systems, they will be described in more detail.

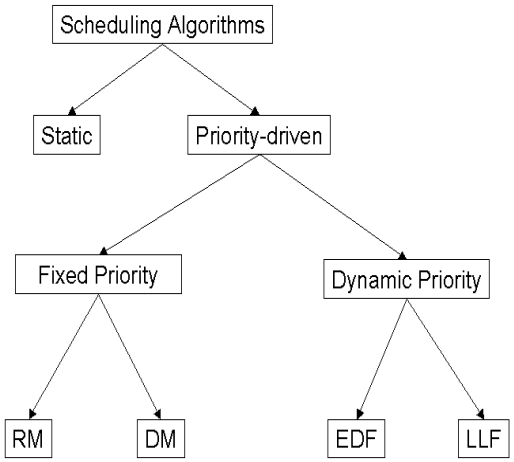

Figure -1. Scheduler's classification

3.1Priority-driven algorithms

3.1.1 Fixed-priority scheduling

The most important scheduling algorithms in this category are Rate Monotonic (RM) [2] and Deadline Monotonic (DM) [4]. The former assigns the higher priority to the task with the shortest period, assuming that periods are equal to deadlines. The latter assigns the highest priority to the task with the shortest deadline. Both algorithms are optimal. Fixed-priority scheduling has been widely studied and the most important real-time operating systems have a fixed-priority scheduler.

3.1.1.1 Schedulability analysis

The first test was provided by Lui and Layland [2]. It is based on the processor utilization of the task set. The total utilization of the set is the sum of the utilizations of all tasks in the set, which is obtained as the quotient of execution time by the period. This utilization is compared with the utilization bound (that depends on the number of tasks). Thus, if the utilization of the set is less or equal to the utilization bound, the task set is schedulable. This schedulability constraint is a sufficient, but not a necessary condition. That is, there are task sets that can be scheduled using a rate monotonic priority algorithm, but which break the utilization bound.

A sufficient and necessary condition was developed by Lehoczky [10] and Audsley [21]. This test is based on the worst case response time of every task. If, in the worst case, a task finishes its execution before its deadline, the task will be schedulable. The worst case response time of a task occurs in the first activation. Moreover, this test is valid for any priority assignation, and it informs not only whether the set is feasible or not, but the task or tasks that miss its deadline.

3.1.2 Dynamic-priority scheduling

Within this category, Earliest Deadline First (EDF) [2] and Least Laxity First (LLF) [13] are the most important. Both are optimal, if any algorithm can find a schedule where all tasks meet their deadline then EDF can meet the deadlines.

In EDF, the highest priority task is the task with the nearest absolute deadline. The absolute deadline is the point in time in which it arrives the deadline of the current activation of the task.

LLF assigns priorities depending on the *laxity*, being the task with the lower laxity, the highest priority task. The term *laxity* refers to the interval between the current time and the deadline, minus the execution time that remains to execute.

Dynamic-priority algorithms have interesting properties when compared to fixed-priority. They achieve high processor utilization, and they can adapt to dynamic environments, where task parameters are unknown. On the contrary, real-time systems community is reluctant to use dynamic-priority algorithms mainly because of the instability in case of overloads. It is also not possible to known what task miss its deadline if the system is not feasible.

3.1.2.1 Schedulability analysis

In [2] it is proved that EDF can guarantee schedulability of tasks when the processor utilization is less than 100%. In this case, deadlines have to be equal than periods, but Dertouzos also proved that EDF is optimal when deadlines are less than periods.

IV. BASIC PARAMETERS THAT AFFECT THE PERFORMANCE IN RTOS [15]

4.1 Multi-tasking and preemptable: To support multiple tasks in real-time applications, an RTOS must be multi-tasking and preemptable. The scheduler should be able to preempt any task in the system and give the resource to the task that needs it most. An RTOS should also handle multiple levels of interrupts to handle multiple priority levels.

4.2 Dynamic deadline identification: In order to achieve preemption, an RTOS should be able to dynamically identify the task with the earliest deadline [19]. To handle deadlines, deadline information may be converted to priority levels that are used for resource allocation. Although such an approach is error prone, nonetheless it is employed for lack of a better solution.

4.3 Predictable synchronization: For multiple threads to communicate among themselves in a timely fashion, predictable inter-task communication and synchronization mechanisms are required. Semantic integrity as well as timeliness constitutes predictability. Predictable synchronization requires compromises.

4.4 Sufficient Priority Levels: When using prioritized task scheduling, the RTOS must have a sufficient number of priority levels, for effective implementation. Priority inversion occurs when a higher priority task must wait on a lower priority task to release a resource and in turn the lower priority task is waiting upon a medium priority task. Two workarounds in dealing with priority inversion, namely priority inheritance and priority ceiling protocols (PCP), need sufficient priority levels.

4.5 Predefined latencies: The timing of system calls must be defined using the following specifications:

• Task switching latency or the time to save the context of a currently executing task and switch to another.

• Interrupt latency or the time elapsed between the execution of the last instruction of the interrupted task and the first instruction of the interrupt handler.

V. ANALYSIS AND OBSERVATIONS

5.1 Scheduling with fixed priority algorithm (RM)

We schedule the following task set with FP (RM priority assignment).

$\tau1 = (1, 4)$, $\tau2 = (2, 6)$, $\tau3 = (3, 8)$.

$U = 1/4 + 2/6 + 3/8 = 23/24$

The utilization is greater than the bound: there is a deadline miss

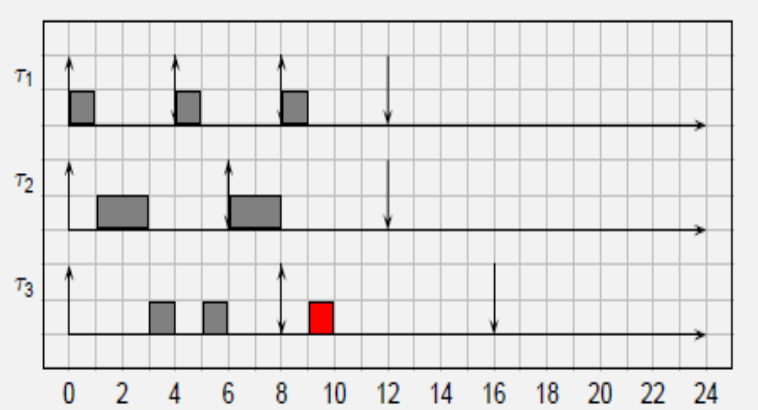

Figure 1. Timeline for RM

Observe that at time 6, even if the deadline of task τ 3 is very close, the scheduler decides to schedule task τ2. This is the main reason why τ3 misses its deadline!

5.2 Scheduling with dynamic priority algorithm (EDF)

Now we schedule the same above task set with EDF.

τ 1 = (1, 4), τ 2 = (2, 6), τ3 = (3, 8).

U = 1/4 + 2/6 + 3/8 = 23/24

Again, the utilization is same. However, no deadline miss in the hyperperiod.

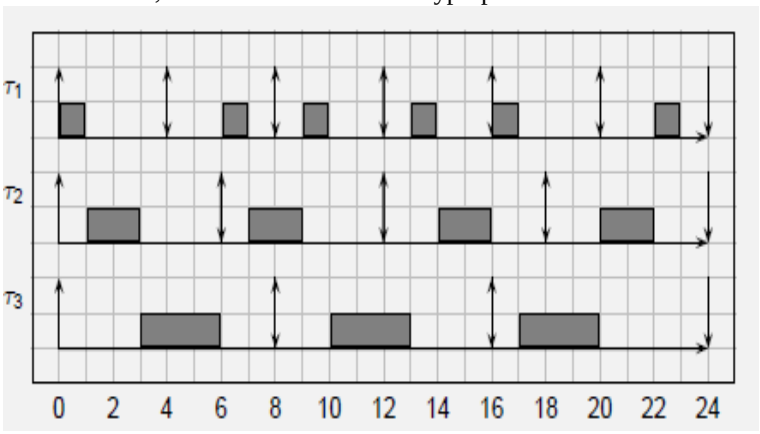

Figure 2. Timeline for EDF

Observe that at time 6, the problem does not appear, as the earliest deadline job (the one of τ 3) is executed. EDF is an optimal algorithm, in the sense that if a task set if schedulable, then it is schedulable by EDF.

EDF can schedule all task sets that can be scheduled by fixed priority, but not vice versa. There is no need to define priorities. In fixed priority, in case of offsets, there is not an optimal priority assignment that is valid for all task sets. In general, EDF has less context switches. In the previous example, the number of context switches in the first interval of time: in particular, at time 4 there is no context switch in EDF, while there is one in fixed priority. As no of context switches are less processor utilization is greater, less idle times.

EDF is less predictable: Looking back at the example, the response time of task τ1: in fixed priority is always constant and minimum; in EDF is variable. If we want to reduce the response time of a task, in fixed priority is only sufficient to give him an higher priority; in EDF we cannot do anything; we have less control over the execution. Fixed priority can be implemented with a very low overhead even on very small hardware platforms (for example, by using only interrupts); EDF instead requires more overhead to be implemented. Computing the response time in EDF is very difficult. EDF is still optimal when relative deadlines are not equal to the periods.

VI. CONCLUSION AND FUTURE SCOPE

From the comparative study it can be concluded that since the concept of "time" is of such importance in real-time application systems, and since these systems typically involve various contending processes, the concept of scheduling is integral to real-time system design and analysis. Scheduling and schedulability analysis enables these guarantees to be provided. From the comparison of real time scheduling algorithms, it is clear that earliest deadline first is the efficient scheduling algorithm if the CPU utilization is not more than 100% but does scale well when the system is overloaded. In the experimental environment, although EDF scheduling algorithm can meet the needs of real-time applications, in practical applications, it is still needed to be improved to meet the evolving needs of real-time systems. In future a new algorithm should be developed which is a mix of fixed and dynamic priority or a scheduling algorithm switch automatically between EDF algorithm and fixed based scheduling algorithm to deal overloaded and under loaded conditions. The new algorithm will be very useful when future workload of the system is changeable.

REFERENCES

[1] Arnoldo Diaz, Ruben Batista and Oskardie Castro, 2013. Realtss: A real-time scheduling simulator. International Conference on Electrical and Electronics Engineering.

[2] C. Liu and James Leyland, January 1973. *Scheduling algorithm for multiprogramming in a hard real-time environment.* Journal of the Association for Computing Machinery, 20(1): 46-61.

[3] C. Yaashuwanth and R. Ramesh, 2010. *Design of real time scheduler simulator and development of modified round robin architecture.* International Journal of Computer Applications.

[4] J.Leung and J. Whitehead, 1982. Performance Evaluation, On the complexity of fixed-priority schedulings of periodic, real-time tasks.

[5] Fengxiang Zhang and Alan Burns, September, 2009. *Schedulability analysis for real-time systems with EDF scheduling.* IEEE Transactions on computers, vol. 58, no. 9.

[6] Hamid Arabnejad and Jorge G. Barbosa, 2013. *List scheduling algorithm for heterogeneous systems by an optimistic cost table.* IEEE.

[7] Luis Bu' rdalo, Andre´s Terrasa, Agustı´n Espinosa, and Ana Garcı´a-Fornes, December, 2012. *Analyzing the Effect of Gain Time on Soft-Task Scheduling Policies in Real-Time Systems.* IEEE Transactions on software engineering, vol. 38, no. 6.

[8] M. Kaladevi and Dr. S. Sathiyabama , 2010. *A comparative study of scheduling algorithms for real time task.* International Journal of Advances in Science and Technology, Vol. 1, No. 4.

[9] Moonju Park and Heemin Park, 2012. *An efficient test method for rate monotonic schedulability.* IEEE.

[10] J. Lehoczky, L. Sha, and Y. Ding, 1989 *The rate monotonic scheduling algorithm: Exact characterization and average case behaviour*. IEEE Real-Time Systems Symposium, 166-171.

[11] N.J. Keeling, April, 1999. *How Priority Inversion messes up real-time performance and how the Priority Ceiling Protocol puts it right.* Real-Time Magazine 99(4): 46-50.

[12] Peng Li and Binoy Ravindran September, 2004. *Fast, Best-Effort Real-Time Scheduling Algorithms.* IEEE Transactions on computers, vol. 53, no. 9.

[13] A.Mok and M.Dertouzos, 1978. Multiprocessor scheduling in a hard real-time environment. 7th Texas Conference on Computing Systems.

[14] R. Le Moigne, O. Pasquier, J-P. Calvez 2004. A Generic RTOS Model for Real-time Systems Simulation with SystemC. IEEE.

[15] S. Baskiyar and N. Meghanathan, 2005. *A Survey On Real Time Systems*. Informatica (29), 233-240.

[16] Shahmil Merchant, Kalpen Dedhia *Performance Comparison of RTOS*.

[17] Sanjay Deshmukh and Dr. N. N. Mhala, 2013. *Comparison of open source rtoss using various performance parameters*. International Journal of Electronics Communication and Computer Engineering Volume 4, Issue (2) REACT-2013, ISSN 2249–071X.

[18] Xin Li1, Zhiping Jia1, Li Ma2, Ruihua Zhang1, Haiyang Wang1, 2009. *Earliest deadline scheduling for continuous queries over data streams*. International Conferences on Embedded Software and Systems

[19] Xiaojie Li and Xianbo He, 2010. *The improved EDF Scheduling Algorithm for Embedded Real-time System in the Uncertain Environment.* ICACTE.

[20] C.D. Locke, 1992. Software architecture for hard real-time applications: cyclic executives vs. priority executives. Journal of Real-Time Systems, 4, 37-53

[21] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings. *Applying a new scheduling theory to static priority preemptive scheduling.* Software Engineering Journal, 5, 284-292,

