Scientia Iranica, Vol. 10, No. 4, pp 471–476

SCIENTIA
IRANICA

Research Note

Establishing Preemption Intelligent Rate-Monotonic Scheduling Algorithm

M. Naghibzadeh* and M. Fathi[1]

Rate-monotonic scheduling algorithm is one of the most widely used scheduling strategies, for real-time systems. Since 1973, when it was first introduced, many researchers have studied its behavior and practical safety verification algorithms have been developed. In this paper, a possible modification to the traditional Rate-Monotonic (RM) algorithm is examined. Namely, by eliminating unnecessary preemption, processor utilization is well improved. Another less important result, due to this modification, is that the number of context switching is decreased, which, in turn, reduces overhead time. The new scheduling strategy is called a preemption Intelligent Rate-Monotonic (IRM) algorithm. It is proved that a system of two tasks is safe if, and only if, the processor load factor is $U \leq 1$. It has also been proved that any system that is safe with the rate-monotonic algorithm is also safe with the IRM algorithm.

INTRODUCTION

Scheduling algorithms play a significant role in the design of safe real-time systems. A real time system must not only be correct but it must also perform its tasks in a timely manner. A hard real-time system is a system in which every request has a deadline before which it must be executed. In such a system missing any deadline may cause a catastrophe, which implies that the cost of missing a deadline is infinitely high [1]. Nuclear power plant and aerospace control systems are examples of hard real-time systems. In a soft real-time system, missing some deadlines will lower the accuracy and/or performance of the system, but the system will still continue to operate. Teleconferencing and multimedia applications are examples of soft real-time systems.

There are two wide classes of scheduling preemptive and non-preemptive. With preemptive schedulers a newly arrived higher priority request can cause to suspend the currently executing lower priority request. A non-preemptive scheduler, on the other hand, executes the current request until completion is reached, even if a higher priority request has arrived, since this request is picked up for execution. Preemptive schedulers usually offer better overall system utilization, so, they are preferred over non-preemptive schedulers. On the other hand, non-preemptive scheduling is usually used in some cases where preemptive scheduling is not practical or the system load is low. Preemptive scheduling is also expensive, since it involves overheads and additional scheduling activities at run time [2,3].

The amount of time given to the system to execute and complete a request is called the deadline parameter of the request. The deadline parameter could be as short as the execution time of the request or it could be much longer. For periodical tasks, both theoretical and industrial attraction is towards systems in which the deadline of a request is at the exact time when a new request is generated from the same task [3-6]. Real-time systems with arbitrary deadline parameters are also studied and interesting results are obtained [1,7].

For static priority scheduling, priorities are assigned to tasks in advance, before any execution begins and the scheduler has to respect these priorities [8]. A request carries the priority of the task, which has generated the request. In static priority scheduling algorithms, whether preemptive or non-preemptive, the decision making process to assign priorities is made during the system design. The dynamic priority

*. *Corresponding Author, Department of Computer Engineering, Ferdowsi University of Mashad, Mashad, I.R. Iran.*

1. *Department of Electrical and Computer Engineering, University of New Mexico, USA.*

assignment is an online activity and the scheduler makes scheduling decisions as the system is running. A dynamic priority preemptive algorithm will decide, on the spot, whether to continue executing the current task or suspend it for another. A dynamic priority non-preemptive algorithm will continue with the current task till completion and then decides which of the ready tasks to be executed next.

Preemptive rate-monotonic scheduling is studied in depth in [3], where Liu and Layland proved that a set of n periodic tasks will always run safe if the least upper bound to processor load factor is $U = n(2^{1/n}-1)$. Sha and Goodenough [9] commented that this bound is pessimistic, because the worst case task set is contrived and unlikely to be encountered in practice.

Many practical issues are dealt with and it is shown that the theoretical results for the RM algorithm are not just an academic curiosity but, also, tools in the design of real systems [9-12]. The RM scheduling of periodic task sets with arbitrary deadlines is studied in [1,7]. The fault-tolerant processing of periodic tasks with rate-monotonic scheduling is studied in [13,14]. In [15], the task switching time for different computers is computed.

In this paper, the concern is with systems running periodic hard real-time tasks, in which each request must be completed before or at the time the next request from the same task is generated. Tasks are independent. Request intervals and execution time for requests from each task are known in advance. Priorities are statically assigned to tasks and all requests from a task have the same priority as the task itself. Preemption is allowed. The scheduling strategy used is a new version of the rate-monotonic strategy, which will be explained later in the next section.

In the rest of the paper, first, the modified version of the traditional rate-monotonic algorithm that is called the preemption Intelligent Rate-Monotonic (IRM) algorithm is introduced. Systems running only two periodic tasks are then studied. Then, it has been theoretically proven that any system that is safe with the RM algorithm is also safe with the IRM algorithm, but, there are many systems that are not safe with the RM algorithm but are safe with the IRM algorithm. A sample is given in Example 1. It is clear that IRM reduces the number of task switching in comparison to the RM algorithm and, hence, reduces the overall system overhead. The exact amount of overhead reduction is left for future research.

PREEMPTION INTELLIGENT RATE-MONOTONIC ALGORITHM

Preemptive scheduling strategy usually offers higher process utilization than its non-preemptive counterpart. For example, in a system running two periodic tasks with a preemptive rate-monotonic algorithm, the least upper bound to processor load factor is approximately 0.83. For a system of n tasks, a feasible schedule on a single processor can definitely be devised when the load factor of the task set is not greater than $n(n^{1/n}-1)$. It converges to 0.69 as n becomes large [3]. The load factor of the set of tasks is defined as:

$$U = \sum_{i=1}^{n} e_i/r_i,$$

where $r_i, i = 1, 2, \cdots, n$, is the request interval of task τ_i and $e_i, i = 1, 2, \cdots, n$, is the execution time of each request made by task τ_i. For a similar system of two tasks with a non-preemptive scheduler to be safe, the two following inequalities must simultaneously hold:

$$\begin{cases} e_1 + e_2 \leq r_1 \\ \lceil r_2/r_1 \rceil . e_1 + e_2 \leq r_2 \end{cases}$$

Considering the case where $r_1 >> e_1$, i.e. r_1 is much greater than $e_1, r_2 >> r_1$ and $e_2 = r_1 - e_1$, the system is safe while process utilization approaches zero. For example, think of a system with $n = 2, r_1 = 5.0, e_1 = 0.0001, r_2 = 499990$ and $e_2 = 4.9999$. In this system, $U = 0.0001/5.0 + 4.9999/499990 = 0.000012$. The system is safe, but any increase in the execution time of either task will make the system unsafe. A similar situation can occur when the number of periodic tasks is greater than two.

It was observed that with the RM algorithm, in certain situations, preemption could be harmful. Consider a system of two periodic tasks with properties $r_1 = 4, e_1 = 2, r_2 = 5$ and $e_2 = 2.1$ and suppose two simultaneous requests are generated from tasks τ_1 and τ_2 at time $t \geq 0$. Under a preemptive rate-monotonic algorithm, the second request by τ_1 at time $t + 4$ will preempt the request generated by task τ_2 at time t. This will cause the request made by task τ_2 at time t to be overrun at time $t + 5$. This situation could be avoided if the running request were not preempted at time $t + 4$.

Consider a situation where a lower priority request is running when a higher priority request arrives. With the RM algorithm, the lower priority request is immediately preempted, no matter what the situation is. It is proposed that if the lower priority request has an earlier, or even equal, deadline than the higher priority request, the lower priority task is not to be preempted. No other change is suggested and otherwise, the algorithm will work as the traditional RM algorithm. The new strategy will be called a preemption Intelligent Rate-Monotonic (IRM) algorithm. The proposed change does not affect the static nature of the scheduling algorithm.

Under the IRM algorithm, when a new request arrives, besides checking to see whether the running

request has a lower priority, as is done in the RM algorithm, it must also check to see if the deadline of the running request is earlier than that of the newly arrived request. The aim is to avoid unnecessary preemption which, as will be seen later, will have a great impact on the load factor of an acceptable set of tasks. On the other hand, it can reduce the overhead time by avoiding unnecessary preemption. However, the latter improvement is not a concern of this paper.

The rest of this paper is concerned with real-time systems running a set of n periodic tasks with a preemption intelligent rate-monotonic scheduling algorithm. The total processor load factor is assumed to be less than, or equal to, one, i.e. $U \leq 1$ and tasks start simultaneously at time zero (or later).

Lemma 1

In a system of two periodic tasks with $U \leq 1$ and the IRM scheduling algorithm, if $t_{2_i}, i = 1, 2, \cdots$, is the end of the ith request interval of τ_2 and t'_{1_i} is the time when the last request from τ_1 is generated prior to time t_{2_i}, then, this request will receive at least $\frac{e_1}{r_1}(t_{2_i} - t'_{1_i})$ execution time within the interval $[t'_{1_i}, t_{2_i}]$.

Proof

Proof is by induction on task two's request interval number. Suppose both tasks start, simultaneously, at time zero. t_{2_1} is the end of task two's first request interval and t'_{1_1} is the time when the last request from τ_1 is generated prior to t_{2_1}. Then, the two following cases can be distinguished.

a) There is no processor idle time within the interval $(0, t_{2_1})$. If the execution time spent on the request made by τ_1 at time t'_1 is less than $\frac{e_1}{r_1}(t_{2_1} - t'_{1_1})$ then (see Figure 1):

$$e_2 + \left(\frac{t'_{1_1}}{r_1}\right).e_1 + \left(\frac{t_{2_1} - t'_{1_1}}{r_1}\right).e_1 > r_2,$$

or:

$$e_2 + \left(\frac{t_{2_1}}{r_1}\right).e_1 > r_2.$$

Dividing both sides by r_2:

$$\frac{e_2}{r_2} + \left(\frac{t_{2_1}}{r_1.r_2}\right).e_1 > 1.$$

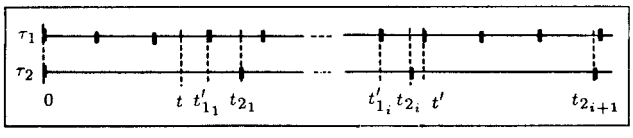

Figure 1. Last processor idle period ends at time t.

Replacing t_{2_1} by r_2 and eliminating r_2 from the nominator and denominator of the second term, it follows that $\frac{e_2}{r_2} + \frac{e_1}{r_1} > 1$, meaning that $U > 1$, which is a contradiction.

b) There is at least one processor idle period within interval $(0, t_{2_1})$. Suppose t is the end of the last processor idle period within that interval. If $t'_{1_1} < t \leq t_{2_1}$, the request that is generated by τ_1 at time t'_{1_1} has consumed e_1 execution time and, since $r_1 \geq t_{2_1} - t'_{1_1}$, by dividing both sides of the inequality by r_1, it follows:

$$1 \geq \frac{t_{2_1} - t'_{1_1}}{r_1}.$$

Multiplying both sides by e_1;

$$e_1 \geq \frac{e_1}{r_1}(t_{2_1} - t'_{1_1}),$$

which again proves that the execution time spent on the request generated by τ_1 at time t'_{1_1} is at least $\frac{e_1}{r_1}(t_{2_1} - t'_{1_1})$.

Now, assume $t \leq t'_{1_1}$. The execution of the request made by τ_2 at time zero must have been finished by time t.

If the execution time spent on the request generated by τ_1 at time t'_{1_1} is less than $\frac{e_1}{r_1}(t_{2_1} - t'_{1_1})$, one can write:

$$\left(\frac{t_{2_1} - t}{r_1}\right).e_1 > (t_{2_1} - t).$$

Dividing both sides by $(t_{2_1} - t)$ one gets $\frac{e_1}{r_1} > 1$, which contradicts $U \leq 1$.

Suppose that the statement is true for the ith request interval of τ_2. It must be proven that the statement is also true for the $(i+1)$th request interval too. Based on the induction hypothesis, the execution time spent on the request made by τ_1 at time t'_{1_i}, where t'_{1_i} is the time at which the last request from τ_1 is generated prior to time t_{2_i}, is greater than, or equal to, $\frac{e_1}{r_1}(t_{2_i} - t'_{1_i})$.

If t' is the time when the first request from τ_1 is made after or at time t_{2_i}, i.e. deadline of the request made at time t'_{1_i}, the execution time needed by the request generated by τ_1 at time t'_{1_i} during the $(i+1)$th request interval of τ_2 is:

$$e' \leq \frac{e_1}{r_1}(t' - t_{2_i}).$$

Therefore, by comparing the first request interval and the $(i+1)$th request interval of τ_2, it can be concluded that:

$$e_2 + e' + \left(\frac{t_{2_{i+1}} - t'}{r_1}\right).e_1 \leq e_2 + \left(\frac{t_{2_1} - 0}{r_1}\right).e_1.$$

Based on the latter inequality, a similar argument, as was given for the first request interval of τ_2, proves that the statement is true for the $(i+1)$th request interval of τ_2.

The following theorem shows the superiority of the IRM algorithm over the RM algorithm, for a two-task real-time system.

Theorem 1

A system of two periodic tasks with a preemption intelligent rate-monotonic algorithm is safe if, and only if, $U \leq 1$.

Proof

First, it is proven that if $U \leq 1$, the system is safe. At the exact time when a request by τ_1 is received, one of the following two situations may occur:

1. The processor is not executing a request from τ_2, whose deadline is earlier than the deadline of the request from τ_1. In this case, if a request from τ_2 is being executed, it will be preempted immediately and the execution of the newly arrived request from τ_1 will be started. On the other hand, if the processor is idle, the newly arrived request from τ_1 will be picked up for execution immediately. In any case, knowing that $U \leq 1$, it is clear that this task will be completed in time.
2. The processor is busy executing a request from τ_2, whose deadline is earlier than the deadline of the request from τ_1. Here, the request under execution cannot be preempted, but it is clear that the newly arrived request from τ_1 will be the last one that can be generated before the deadline for the request from τ_2 is over. Suppose that the request from τ_1 is arrived at time t'_{1_i} and the deadline for the request from τ_2 will be over at $t_{2_i} > t'_{1_i}$, see Figure 2.

According to Lemma 1, the execution time consumed by the request from τ_1, up to time t_{2_i}, will be at least $\frac{e_1}{r_1}.(t_{2_i} - t'_{1_i})$.

This task can receive $t''_{1_{i+1}} - t_{2_i}$ execution time after time t_{2_i}, where $t''_{1_{i+1}}$ is the deadline for the request that is generated at time t'_{1_i} from τ_1. Therefore, the total possible execution time, which can be spent on the request from τ_1, which is generated at time t'_{1_i}, is:

$$\frac{e_1}{r_1}.(t_{2_i} - t'_{1_i}) + (t''_{1_{i+1}} - t_{2_i}).$$

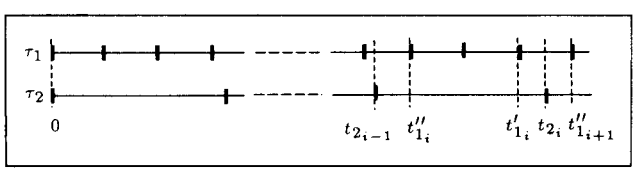

Figure 2. A two-task system with $U \leq 1$.

Since $\frac{e_1}{r_1} \leq 1$, one has:

$$\begin{aligned}\frac{e_1}{r_1}.(t_{2_i} - t'_{1_i}) + (t''_{1_{i+1}} - t_{2_i}) &\geq \frac{e_1}{r_1}(t_{2_i} - t'_{1_i}) \\ &+ \frac{e_1}{r_1}(t''_{1_{i+1}} - t_{2_i}) \\ &\geq \frac{e_1}{r_1}(t''_{1_{i+1}} - t_{1_i}) \\ &\geq e_1.\end{aligned}$$

Therefore, every request from τ_1 will be executed in time.

Within ith, $i = 1, 2, 3, \cdots$, request interval of task τ_2, there is a request from τ_1 that is generated prior to, or at the time the ith request from τ_2 is generated. If the deadline for this request is at time t''_{1_i}, according to Lemma 1, the execution time needed by this request within the interval (t_{2_i-1}, t''_{1_i}) is $e' \leq \frac{e_1}{r_1}(t''_{1_i} - t_{2_{i-1}})$.

Therefore, in order for the ith, $i = 1, 2, \cdots$, request from τ_2 to be executed in time, one must have:

$$e' + \lfloor (t_{2_i} - t''_{1_i})/r_1 \rfloor . e_1 + e_2 \leq r_2.$$

But,

$$\begin{aligned}e' + \lfloor (t_{2_i} - t''_{1_i})/r_1 \rfloor . e_1 + e_2 &\leq \frac{e_1}{r_1}(t''_{1_i} - t_{2_{i-1}}) \\ &+ (t_{2_i} - t''_{1_i}).\frac{e_1}{r_1} + e_2, \\ &\leq \frac{e_1}{r_1}.r_2 + e_2.\end{aligned}$$

Since $U = \frac{e_1}{r_1} + \frac{e_2}{r_2} \leq 1$, therefore, $\frac{e_1}{r_1}.r_2 + e_2 \leq r_2$.

It is clear that if $U > 1$, the system is not safe.

Example 1

Consider a system of two tasks with $r_1 = 8.0, e_1 = 1.9, r_2 = 9.9$ and $e_2 = 6.11$. The processor load factor for this system is $U = \frac{e_1}{r_1} + \frac{e_2}{r_2} = \frac{1.9}{8.0} + \frac{6.11}{9.9} = 0.855$. This system is not safe with the RM algorithm and when the system starts at time zero, the first request from τ_2 will not be executed in time. This system will run safe with the IRM algorithm. The system will be safe even if, for example, e_2 is increased to become 7.5. The processor load factor is now:

$$\frac{e_1}{r_1} + \frac{e_2}{r_2} = \frac{1.9}{8.0} + \frac{7.5}{9.9} = 0.9951.$$

Theorem 2

Any system that is safe with the RM algorithm is also safe with the IRM algorithm.

Proof

Suppose the statement is not true and there exists a system consisting of n periodic tasks which is running safe with the RM algorithm, but not with the IRM algorithm.

When the system starts at time zero (or any other time), there must be a request from task $\tau_i, i = 1, 2, \cdots, n$, that will not be executed by the time its deadline is reached. If this request is generated at time t, its deadline will be at time $t_1 = t + r_i$. When the IRM algorithm is used, there might be some processor idle time within the interval $(0, t_1)$. The statement will be proven later when there is no processor idle time within the interval $(0, t_1)$.

Suppose that the first processor idle time starts at time t_2 and ends at time $t_3 > t_2$ (see Figure 3). It will be shown that the processor will also be idle in the interval (t_2, t_3) when the RM algorithm is used. Since $t_2 < t$ and the system is safe with the RM, all requests generated in the interval $[0, t_2]$ are executed in time by both the RM and the IRM algorithms. On the other hand, no request is generated in the interval (t_2, t_3). Therefore, the processor must be idle within the interval (t_2, t_3) with the RM algorithm. Using a similar argument, if there are other processor idle times prior to t with the IRM algorithm, the processor will be idle within the same intervals with the RM algorithm too.

Suppose, the last processor idle time ends at $t_4 < t$. With the IRM algorithm and within the interval $[t_4, t_1]$, no request from tasks $\tau_{i+1}, \tau_{i+2}, \cdots, \tau_n$, with deadline at time t_1 or later, will have higher priority than the request generated at time t by τ_i. Requests from $\tau_{i+1}, \tau_{i+2}, \cdots, \tau_n$, with deadlines prior to time t_1, will have to be completed before time t_1, under either the RM or the IRM algorithms. All requests made by $\tau_1, \tau_2, \cdots, \tau_{i-1}$, within the interval (t_4, t_1), have higher priorities over the request made by τ_i at time t, when the RM algorithm is used. However, there might exist circumstances that the request generated by τ_i at time t, is given higher priority over some requests made by $\tau_1, \tau_2, \cdots, \tau_{i-1}$, within the interval (t_4, t_1) with the IRM algorithm. Therefore, when RM is used, there must exist a request from some task that will overrun prior to, or at time, t_1.

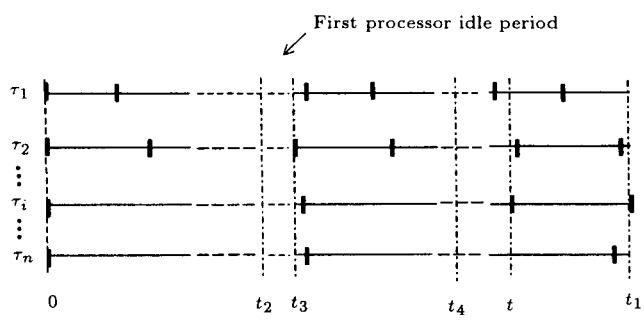

Figure 3. A request from τ_i is not executed in time.

The argument for the situation when there is no processor idle time within the interval $(0, t_1)$ is similar to the argument given above for the interval (t_4, t_1), with t_4 being replaced by 0.

This contradicts the assumption that the system is safe with the RM algorithm and not safe with the IRM algorithm.

Based on Theorem 2, all safety verification algorithms for the RM scheduling strategy can also be used for the IRM algorithm [3-6,8,10].

Example 1 shows that not all systems that run safe with the IRM algorithm can also run safe with the RM algorithm.

CONCLUSION

In this paper, a modified version of the rate-monotonic algorithm, which is called an intelligent rate-monotonic algorithm, has been introduced. The performance of the two algorithms is compared and it was shown that the performance of an intelligent rate-monotonic algorithm is superior to a rate-monotonic algorithm in two directions, without noticeable increase in complexity. First, there are systems that are safe with the IRM and are unsafe with RM. Second, it reduces the number of task switching, by its nature, and increases total system performance. The authors believe that IRM could substitute RM in all practical areas where RM is previously used. Future research could concentrate on detailed comparison of the two algorithms for real applications.

REFERENCES

1. Tindell, K., Burns, A. and Wellings, A.J. "An extendible approach for analyzing fixed priority hard real-time tasks", *Real-Time Systems*, **6**(2) pp 133-151 (1994).
2. Kim, K.H. and Naghibzadeh, M. "Prevention of task overruns in real-time non-preemptive multiprogramming systems", *Proceedings of ACM Performance Evaluation Review*, pp 267-276 (Summer 1980).
3. Liu, C.L. and Layland, J.W. "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the ACM*, **20**(1), pp 46-61 (1973).
4. Lehoczky, J.P., Sha, L. and Ding, Y. "The rate-monotonic scheduling algorithm: Exact characterization and average case behavior", In *Proceedings of the 10th Real-Time Systems Symposium*, Santa Monica, CA (Dec. 1989).
5. Manabe, Y. and Aoyagi, S. "A feasibility decision algorithm for rate-monotonic scheduling of periodic real-time tasks", *IEEE 1st Real-Time Technology and Applications Symp.*, pp 212-218 (May 1995).

6. Manabe, Y. and Aoyagi, S. "A feasibility decision algorithm for rate-monotonic and deadline monotonic scheduling", *Real-Time Systems*, **14**(2), pp 171-181, (Mar. 1998).
7. Lehoczky, J.P. "Fixed priority scheduling of periodic task set with arbitrary deadlines", *Proc. Real-Time Systems Symposium*, pp 201-209 (1990).
8. Leung, J. and Whitehead, J. "On the complexity of fixed priority scheduling of periodic real-time tasks", *Performance Evaluation*, **2** (1982).
9. Sha, L. and Goodenough, J.B. "Real-time scheduling theory and Ada", *IEEE Computer*, pp 53-62 (April 1990).
10. Klein, M., Ralya, T., Pollak, B., Obenza, R. and Harbour, M.G., *A Practitioners Handbook for Real-Time Analysis: Guide to Rate-Monotonic Analysis of Real-Time Systems*, Kulwer Academic Publisher (1993).
11. Naghibzadeh, M. "Safety verification of real-time computing systems serving periodic devices", *Journal of Engineering, IRI*, **7**(3) (Aug. 1994).
12. Sha, L., Rajkumar, R. and Sathaye, S.S. "Generalized rate-monotonic scheduling theory: A framework for developing real-time systems", in *Proceeding of the IEEE*, **82**(1), pp 68-82 (Jan. 1994).
13. Ghosh, S., Melhem, R. and Mosse, D. "Fault-tolerant rate-monotonic scheduling", *Journal of Real-Time Systems*, **15**(2) (Sept. 1998).
14. Pandya, M. and Malek, M. "Minimum achievable utilization for fault-tolerant processing of periodic tasks", *IEEE Transactions on Computers*, **47**(10) (1998).
15. Otterbach, R. and Leinfellner, R. "Real-time simulation: Requirements and the state of the technology", Translated from *Virtuelles Ausprobieren Elektronik*, **8** (1999).