A Scheduling of Hard-Aperiodic Request in Dynamic Priority Systems

Hyungill Kim[9], Sungyoung Lee[9], and Jongwon Lee[99]

Department of Computer Engineering, Kyung Hee University, Seoul, Korea[9]
Software Research Lab., Korea Telecom, Seoul, Korea[99]

ABSTRACT

In this paper, we discuss the problem of jointly scheduling both hard deadline of periodic and aperiodic tasks in dynamic priority systems. The proposed scheduling scheme has extended the APS (Alternative Priority Scheduling) algorithm which is developed by the authors. The APS algorithm has a simple slcak calculation method which in consequence makes it be practical. The paper develops an efficient acceptance test method for hard-aperiodic requests. The on-line acceptance test should be performed to determine whether the timing requirements of the arriving hard aperiodic task can be met while guaranteeing all the deadlines of periodic tasks and any already accepted but not yet completed aperiodic tasks.

1. Introduction

In the last few years considerable researches in the areas of jointly scheduling of both hard-deadline of periodic and aperiodic tasks have been done either in fixed-priority or dynamic priority systems. A number of algorithms that solve this problem in fixed-priority systems can be found in the literature [LEHOCZKY 95], [SHIN]. Thuel and Lehoczky [LEHOCZKY 95] have developed an extension of the slack-stealing algorithm [LEHOCZKY 92], which provides the largest amount of processing capacity for aperiodic tasks subject to guaranteeing the deadlines of the periodic tasks, in fixed priority-systems. The algorithm tests acceptance for hard aperiodic tasks for guaranteeing tasks at any priority level while it assumes that the periodic deadlines must all be met. Shin and Chang [SHIN] proposed an elegant aperiodic task scheduling, called the Reservation-Based (RB) algorithm which can guarantee all periodic-task deadlines while minimizing the probability of missing aperiodic task deadlines. The RB algorithm reserves a fraction R_s of processor time in each unit cycle, which is defined as the greatest common divisor of all task periodics, for the executing aperiodic tasks without missing any periodic task deadline. They also formally derive the value of R_s that maximizes the processor time reservable for the execution of aperiodic tasks without missing any periodic task deadline. However, the RB algorithm has some limitations. First, if the length of unit cycle is big enough, then it is hard to predict the success ratio of the aperiodic tasks even though the value of R_s is large. Second, it assumes that the relation between R_s and the probability of guaranteeing aperiodic tasks is establish for the case when the execution time of aperiodic tasks is exponentially distributed. Moreover, RB algorithm assumes that the probability of guaranteeing an aperiodic task is a monotonic increasing function of R_s which is not all the true in real world.

Less attention has been made to the same problem in the context of dynamic priority systems.

Work on the on-line scheduling of hard aperiodic tasks in dynamic priority methods has been reported by Chetto and Chetto [CHETTO], and Schwan and Zhou [SCHWAN]. Their work assumes that all periodic tasks are scheduled according to the Earliest Deadline algorithm [LIU]. Especially a point to note is that Schwan and Zhou's algorithm does not give any preferential treatment to the periodic tasks, unlike common approaches to soft-aperiodic tasks in fixed-priority preemptive systems. Every task is subject to an acceptance-rejection test upon arrival. The algorithm, however, may lead to an undesirable implementation overhead if the real-time workload is mainly periodic. Tia [TIA] introduced an optimal scheduling, called algorithm *SD* (for Sporadic, Dynamic) which has linear time complexity, of aperiodic requests in dynamic priority systems based on the slack stealing approach. The main idea of their algorithm is to partition the periodic requests in sets such that the slack of all the requests in the same set is affected by the same scheduling events. Spuri, Buttazzo, and Sensini [SPURI] have introduced Total Bandwidth Server based on the integration of an efficient aperiodic server and technique including a rejection and a reclaiming strategies. This approach, however, still has a remaining problem; how to reserve an optimal bandwidth preservation level.

This paper discusses the problem of jointly scheduling both hard deadline periodic and aperiodic tasks using dynamic priority systems. As per Lehoczky [LEHOCZKY 95], hard deadline aperiodic tasks are special importance since they alert conditions or from failures of hard deadline periodic tasks which fail the responsibility checks to validate their results and must be retried and completed before the original deadline elapses. When not all of the timing requirements of the periodic and aperiodic tasks can be met simultaneously, we have to choose as to which tasks to accept for processing. Generally, the on-line acceptance test should be performed to determine whether the timing requirements of the arriving hard aperiodic task can be met while guaranteeing all the deadlines of periodic tasks and any already accepted but not yet completed aperiodic tasks. In this paper we present a simple acceptance test mechanism for hard aperiodic requests which is an extension of the APS algorithm [KIM].

The APS algorithm developed by the authors has adopted a new type of scheduling scheme which chooses either an EDF or a CEF (Critical Execution time First) scheduling policy alternatively for a given periodic task set at on-line while referencing information in the off-line built CTI (Critical Task Indicating) table [LEE]. The CTI table is created by the ***deadlinewise preassignment*** scheduling policy in which tasks are preassigned toward deadlines at their maximum. The scheduling information on the CTI table enables a scheduler to have a predictability and reduces the overall computational complexity at run time. Building a CTI table is very similar to the reverse schedule of the EDF introduced by Chetto and Chetto [CHETTO]. However, there are differences between the two approaches in terms of their applications and pursuing goals which are shown in [LEE]. Recall that the CTI algorithm considers the problem of the joint scheduling of periodic and aperiodic tasks. The algorithm basically performs the on-line scheduling based on either a fixed-priority or a dynamic-priority scheduling while referencing the scheduling information on the off-line built CTI table.

The CEF scheduling gives the highest priority to a periodic task which has the largest amount of

critical task's execution time (cet) at on-line. The ***critical task*** is a periodic task which must be scheduled at certain time ***t***, otherwise its deadline should not be guaranteed. The ***cet*** is the time differences between all the computation requirement (***$Cr_i(t)$***) and all the computation processing completed (***$Cp_i(t)$***) on the CTI table for each periodic task until time ***t***: (***$Cr_i(t)$-$Cp_i(t)$***). The ***cet*** is used to evaluate whether the preassigned periodic tasks in the CTI table must be executed or not to meet their deadlines. The major reason for introducing an alternative scheduling policy is to get a fast response time for aperiodic requests.

The APS algorithm introduced the concept of an ***optimistic*** slack calculation scheme rather than a ***minimal*** method which is used in the slack-stealing based schedulings. The term ***minimal*** method means that the slack stealing algorithm [LEHOCZKY 92] uses the actual slack as the minimum slack value of all periodic requests to ensure that all the lower priority periodic deadlines are met. On the other hands, the term ***optimistic*** means that a scheduler regards all the scheduling time as the slack except for the execution time of a given periodic task set to guarantee their deadlines. The ***optimistic*** approach therefore, is to maximize the reclaiming of unused computation time and not necessarily to search the slack in all the levels of periodic tasks. Thus, the computational overhead to calculate slack can be reduced.

An acceptance test of the APS algorithm is to perform when an aperiodic task arrives it determines whether there is enough time available during the interval between the arrival time and the deadline to complete the execution while ensuring that all the periodic tasks as well as previously accepted aperiodic tasks meet their deadlines. This acceptance test mechanism is to merely modify the ***optimistic*** slack calculation scheme to serve the hard-aperiodic requests.

The remainder of this paper is organized as follows. Section 2 establishes a system model and assumptions used in this paper. Section 3 describes the ***optimistic*** slack calculation approach and discussion, shows an acceptance test method, explains operation of the APS algorithm, and gives an example of the algorithm. Finally, we conclude the paper in section 4.

2. Task Model

Consider a uniprocessor real-time system with a set **T** of n independent, preemptable periodic tasks, (τ_1, τ_2, ..., τ_n). Each task, τ_i , has a worst-case computation requirement C_i, a period T_i, an initiation time or an offset ϕ_i relative to some time origin, and a deadline d_i. Hence, we denote the system $\mathbf{T} = \{\tau_i(C_i,T_i,d_i): 1 \le i \le n\}$. In response to external events which occur at random time instants, the aperiodic tasks, $\{J_k, k \ge 1\}$ are introduced. Each hard aperiodic request $J_i(a_i, e_{i, Di})$ is characterized by its arrival time a_i, its worst-case execution time e_i, and its deadline D_i. Let hyperperiod, ***H***, be the least common multiple of all the periodic task's periods. For simplicity, the algorithm uses the following assumptions.

(A1) Deadline for a periodic task's instance is equal to the next request of the task.

(A2) $T_i = d_i$ and all initiation times or relative offsets $\{\phi_i, 1 \le i \le n\}$ are synchronized to 0.

(A3) Preemption over a periodic or an aperiodic task is always possible

(A4) All overhead for context switching is counted into the corresponding periodic or aperiodic task's computation requirements.

(A5) The aperiodic task sequence is not known in advance, however, its execution time and deadline as are known when an aperiodic tasks arrives

3. Alternative Priority Scheduling

3.1 Slack Calculation

The major motivation of the APS algorithm lies how to calculate the slack in a simple manner. In the conventional joint schedulings of aperiodic and periodic tasks, the scheduling time interval **[0, *t*]** is filled up by the execution time of periodic task instances P_{ij} (t), the execution time for aperiodic requests $A(t)$, and the idle time $I(t)$. The formula (1) depicts this concept.

$$t = \sum_{i,j} P_{ij}(t) + A(t) + I(t) \quad \cdots \quad (1)$$

The slack within **[0, *t*]**, denoted by $S_0(t)$, is generally defined as an available time for the aperiodic requests while guaranteeing the deadlines of the given periodic requests. Thus, $S_0(t)$ is a total amount of time from 0 to t except for the sum of the execution time of periodic requests. The actual slack for aperiodic requests, however, is less than or equal to $S_0(t)$ because it includes idle time (see formula (2)).

$$S_0(t) = A_0(t) + I_0(t) \quad \cdots \quad (2)$$

The slack stealing algorithm [LEHOCZKY 92], [LEHOCZKY 95] uses the actual slack as the minimum slack value of all periodic requests to ensure that all the lower priority periodic deadlines are met (see formula (3)).

$$S^*(t) = min_{\{1 \le i \le n\}} S_i(t) \quad \cdots \quad (3)$$

On the contrary, in the APS algorithm, we have newly introduced the concept of an optimistic slack calculation method to get a maximum slack $S(t)$ at current time t. The idea of the optimistic approach is that a scheduler regards whole of the scheduling time as a slack except for the execution time of the given periodic tasks to meet their deadlines. Thus, the optimistic approach is not necessarily to calculate slack at all the levels of periodic tasks which in turn reduces the computational complexity to calculate slack.

$$S(t_1, t_2) = t_2 - t_1 - \sum_{i=1}^{n} \delta_i \quad \cdots \quad (4)$$

The formula (4), which shows a way of an optimistic slack calculation, demonstrates an actual value of slack ($S(t_1, t_2)$) in **[t_1, t_2]**. The $S(t_1, t_2)$ can be obtained by subtracting the minimum amount of the execution time of periodic tasks ($\sum_{i=1}^{n} \delta_i$) from (**t_2 -t_1**).

In order to calculate slack efficiently, we can decompose a hyperperiod into a set of scheduling

zones, denoted by **[*t, Z(t)*]** where *Z(t)* can be calculated by the formula (5). The meaning of **[*t, Z(t)*]** is a time interval from ***t*** to the earliest deadline of a periodic task. The reason for introducing the decomposition strategy is to perform an efficient slack calculation by using *Z(t)* as a scheduling checkpoint. For example, when an aperiodic request happens at any arbitrary time ***t***, the foremost concern is how to schedule that aperiodic task while guaranteeing the very following deadline of periodic task's instance. By this way, we can easily evaluate the slack availability while considering the deadlines of periodic tasks. Notice that *Z(t)* can be obtained from the sequences of periodic tasks' instances in the CTI table.

$$Z(t) = min\{d_{ij} | 1 \le i \le n, j = \left\lceil \frac{t}{T_i} \right\rceil\} \cdots (5)$$

Figure 1 depicts an example of how to obtain Z(t) for the given three periodic tasks $\tau_1(1,5,5)$, $\tau_2(1,7,7)$, $\tau_3(3,10,10)$ where the parameters are the worst case computation time, deadline, and period of the tasks, respectively.

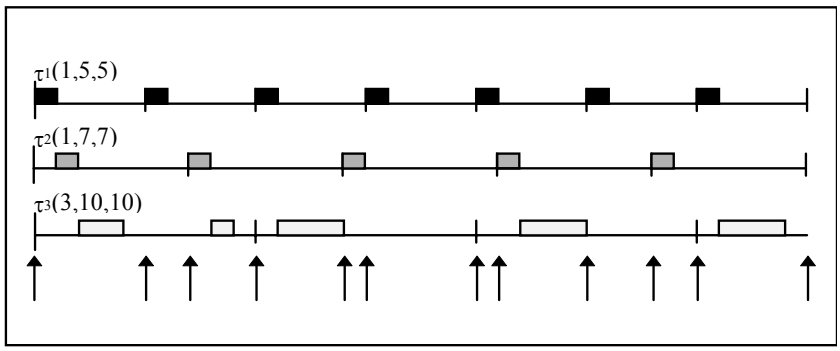

Figure 1. Example of how to decide *Z(t)*

Consequently, *Z(t)* is a parameter to calculate slack from current time ***t*** to the earliest deadline of the periodic task. Thus, we can calculate the slack as follows:

$$S(t) = S(t, Z(t)) = Z(t) - t - \sum_{i=1}^{n} \delta_i \cdots (6)$$

In the formula (6) which can be easily induced from the formula (4), we call δ_i as the ***cet*** and it can be obtained from the CTI table. Recall that the ***cet*** is used to evaluate whether the preassigned periodic tasks in the CTI table should be executed or not to meet their deadlines. To clarify the term ***cet***, we introduce two notations; one is a cumulating all the computation processing completed (***$Cp_i(t)$***) and the other is a cumulating all the computation requirement (***$Cr_i(Z(t))$***) in the CTI table. Hence, we can formally define the ***cet*** as follows:

$$\delta_i = \begin{cases} 0, & \text{if } (Cr_i(Z(t)) - Cp_i(t)) < 0 \\ (Cr_i(Z(t) - Cp_i(t)), & \text{otherwise} \end{cases} \cdots (7)$$

For simple notation, we define the sum of ***cet***s at time ***t*** as follow:

$$\Delta(t) = \sum_{i=1}^{n} \delta_i \qquad \cdots \ (8)$$

3.2 Discussion

As we mentioned before, the APS algorithm chooses either an EDF or a CEF scheduling alternatively at on-line for a given periodic task set based on $\Delta(t)$ while referencing information on the off-line built CTI table. The major reason to introduce an ***alternative scheduling*** policy is to get fast response time for aperiodic requests. Consequently, the APS algorithm can find more slack which in turn may increase an acceptance ratio of the hard-aperiodic tasks. Suppose there is no aperiodic request. At that time, if the scheduler performs on-line scheduling using EDF only, then some periodic tasks (***non-critical tasks***) of which deadlines are earlier than those of the ***critical tasks*** may be scheduled first (in case of a deadline of a non-critical task is earlier than that of a ***critical task***.) In this case, the scheduler finds no more slack available since the value of slack obtained from formula (6) turns to be negative.

Note that a ***critical task*** does not mean a periodic task which has the earliest deadline in a given periodic task set. Therefore, as the value of Δ(t) is increased, in consequence, the value of slack is decreased. Consequently, an aperiodic task may not be scheduled under EDF policy even though the slack is available. On the other hand, in the CEF algorithm, since the scheduler allocates the minimum amount of execution of periodic tasks within *Z(t)* based on the information in the off-line built CTI table, the slack value obtained from the formula (6) will never be a negative which in turn has higher probability to find more slack. Recall that we define that the CEF scheduling gives the highest priority to the periodic task which has the largest value of the ***cet***. Thus, if there is periodic task(s) in the ready queue and Δ(t) > 0, then service the periodic task with the CEF scheduling policy. Obviously, we can get the Δ(t) from information on the CTI table. In other words, if ***cet*** > 0, then the scheduler must follow the information on the CTI table that instructs the scheduling (assignment sequence) of periodic tasks.

3.3 Acceptance Test

An acceptance test of the APS algorithm is to perform that when a hard aperiodic task arrives the scheduler determines whether there is sufficient time available during the interval between the arrival time and the deadline to complete the execution while ensuring that all the periodic tasks as well as previously accepted aperiodic tasks meet their deadlines. An available time for a hard aperiodic request in $[t_1, t_2]$ is as follows.

$$S(t_1, t_2) = | t_1 - t_2 | - \Delta(t_1, t_2) \ \ldots (9)$$

An available time for aperiodic tasks, called the slack, during the interval between the current time *t* and the deadline of hard aperiodic task (*D*) is dependent on the periodic tasks' execution time

during that interval. Consequently, the available slack ($S(t_1, t_2)$), can be calculated by subtracting the ***cet*** from the time interval [t_1, t_2]. From the formula (9), we can get the $\Delta(t_1, t_2)$ using the definition $\Delta(t)$ which is appeared on the formula (8) as follow (formula 10).

$$\Delta(t_1, t_2) = \sum_{i=1}^{n} \delta(t_1, t_2) = \sum_{i=1}^{n} C_r(t_2) - C_p(t_2) \ldots (10)$$

Meanwhile, we define the surplus slack, γ_i, in the formula (11). The γ_i means that the surplus time for aperiodic requests, which can be calculated by subtracting sum of the ***cet*** and execution time of all accepted aperiodic tasks from the scheduling zone (D_i-t).

$$\gamma_i = \gamma_{i-1} + D_i - D_{i-1} - \Delta(D_{i-1}, D_i) - e_i \ldots (11)$$
where, $D_0 = t$, $\gamma_0 = 0$, $D_i \geq D_{i-1}$

Figure 2 shows an acceptance test routine of the APS algorithm for a newly arrived hard-aperiodic task at time t. We assume that D(i), e(i) is corresponding to the deadline, the execution time of ith aperiodic task, respectively.

```
1  surplus_slack = 0, D(0)=t, Δ = 0, j=1, γ(0) = 0, n = number of periodic tasks, m = number of
   hard aperiodic tasks for acceptance test;
2  for(j = 1; j <= m; j++) {
3      for ( i = 1; i <= n; i++)    Δ += MAX(Cr(D(j)) - Cp(D(j)), 0);
4          γ(j) = γ(j-1) + D(j) - D(j-1) - Δ - e(j);
5      if (γ(j) < 0) return Reject;
6  } return Accept;
```

Figure 2. The pseudo-code of acceptance test mechanism in the APS algorithm

In Figure 2, initially a surplus slack is set to zero, D(0) is set to t, $\gamma(0)$ is set to 0. At line 2, we are doing an acceptance test for all aperiodic task until no available slack is remained. At line 3, we can get Δ (called ***cet***) during the time interval between the current time t and the D. At line 4, a surplus slack is calculated by the formula (11). At line 5 and 6, if the value of surplus slack is less than zero then the hard aperiodic task j is rejected, otherwise it is accepted. Note that once an aperiodic task is accepted it can not be lost during its execution time.

3.4 Operation of the Algorithm

In this subsection, we illustrate an operation of the APS algorithm. Figure 3 shows a pseudocode of the APS algorithm. At line 1 and 2, the off-line CTI table is created and the

scheduling parameters are initialized. At line 4, the scheduler calculates slack. From line 6 to 8, when a hard aperiodic task arrives, an acceptance test routine will be invoked. If the acceptance test routine returns a result as Accept, then the newly arrived hard aperiodic task is enqueued for service, otherwise it is discarded. From line 9 to 12, an aperiodic task is serviced if slack is available and there are aperiodic tasks in the queue. At line 13 and 14, if there are periodic tasks, the algorithm alternatively chooses either an EDF or a CEF scheduling based on $\Delta(t)$. At line 22, if there is no periodic and aperiodic task to be serviced, the scheduler is idle. Note that the computational complexity of the APS off-line approach is $O(n\log_2 n)$ while that of the on-line acceptance test is $O(n)$ where n is the number of aperiodic tasks.

```
 1  Build off-line CTI table
 2  Initialize counters
 3  while TRUE do
 4     Calculate S(t) and Cri(Z(t));
 5     while (t < Z(t)) do
 6     if (a hard aperiodic task arrive) then
 7          if ((A(t) is Accept) then add the hard aperiodic task to the aperiodic queue
 8     else Reject the hard aperiodic task
 9     if (there is slack and there is aperiodic task in the queue) then
10          while (S(t) > 0 and there is aperiodic task in the queue) do
11             Service the aperiodic task;
12            Update S(t)       /* reduce the amount of service time from S(t) */
13     else if (there is periodic task in the queue) then
14        if ( Δ (t)>0) then
15                 Service the periodic task with the CEF;
16              Update Cpi(t)     /* add the amount of processing time */
17        else
18              Service the periodic task with the EDF;
19              Update Cpi(t)     /* add the amount of processing time */
20              Update S(t)       /* reduce the amount of processing time */
21     else
22        Process idle state;
23        Update S(t)    /* reduce the amount of idle time */
24     endwhile
25  endwhile
```

Figure 3. A pseudocode of the Hard-APS algorithm

3.5 An Example

In this subsection, we demonstrate an example of scheduling using the APS algorithm. Suppose there are three periodic tasks $\tau_1(1,5,5)$, $\tau_2(1,7,7)$, $\tau_3(3,10,10)$ where the parameters are the worst case computation time, deadline, and period of the tasks, respectively. We assume that the

first hard aperiodic task J_1 arrives at a_1=4 with its execution time e_1=5 and deadline D_1=19 (Figure 4).

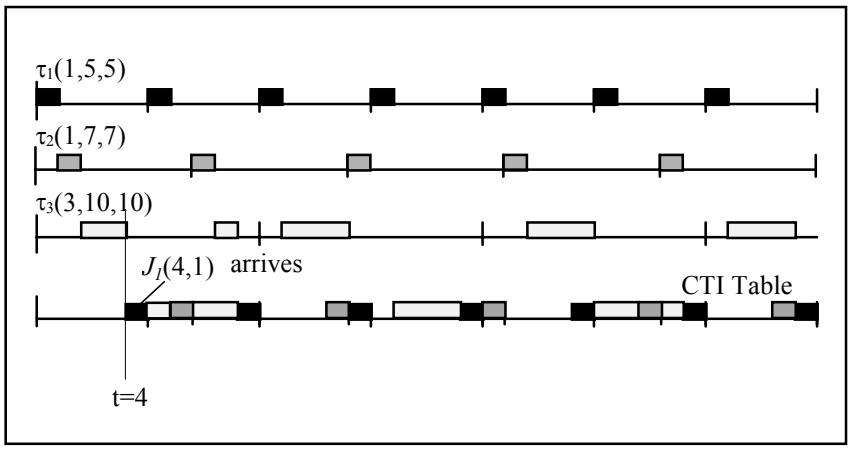

Figure 4. An example of the APS algorithm

Then, we can calculate the slack $S(0)$ as follow:

$$S(0) = S(0, Z(0)) = S(0,5) = 5 - 0 - \sum_{i=1}^{3} \delta_i$$
$$= 5 - (1 + 0 + 0)$$
$$= 4$$

Thus, the available slack within **[0, 5]** becomes 4 units. Since there is no aperiodic task at t=0, the critical task τ_{11} is invoked by the CEF algorithm. From t=1 to t=4, τ_{21} and τ_{31} are invoked by the EDF algorithm and 3 units of the slack value are consumed which means S(4)=1. At time t=4, the aperiodic task J_1 arrives. Then the algorithm performs an acceptance test. S(4,19) can be calculated as follows;

$$S(4,19) = 19 - 4 - \sum_{i=1}^{3} \delta_i(t)$$
$$= 15 - (3 + 1 + 4)$$
$$= 7$$

As a result, the hard-aperiodic task J_1 is accepted since the amount of available slack (7) upto its deadline 19 is larger than the amount of execution time of J_1 (5).

4. Summary

In this paper, we introduced an efficient acceptance-rejection test mechanism for the hard-aperiodic requests based on the APS scheduling policy which is has simple slack calculation scheme. An acceptance test of the APS algorithm is to perform when an aperiodic task arrives it

determines whether there is enough time available during the interval between the arrival time and the deadline to complete the execution while ensuring that all the periodic tasks as well as previously accepted aperiodic tasks meet their deadlines. This acceptance test mechanism is to merely apply the ***optimistic*** slack calculation approach to serve the hard-aperiodic requests.

When developing the scheduling algorithms, we should consider the practical applications in real-world. The APS algorithm, in this respect, does satisfy the practicability and predictability. By introducing an ***optimistic*** slack calculation method, we can achieve the implementation simplicity. By using the off-line CTI scheduling policy, we can grasp the scheduling predictability. Our ongoing work is to enhance the algorithm to be more robust in the transient overload and to compare our simulation results to that of Tia [TIA] and Spuri [SPURI].

References

[CHETTO] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, Vol. 15, No. 10, pp. 466-473, 1989.

[DAVIS] R. Davis and A. Wellings, "Dual Priority Scheduling", Proceedings of the IEEE Real-Time Systems Symposium, pp. 100-109, December 1995.

[HOMAYOUN] N. Homayoun and P. Ramanathan, "Dynamic Priority Scheduling of Periodic and Aperiodic Tasks in Hard Real-Time Systems", Real-Time Systems: The International Journal of Time-Critical Computing Systems, Vol. 6, No. 2, pp. 207-232, 1994

[KIM] H. Kim, S. Lee and J. Lee, "Alternative Priority Scheduling Algorithm: A Soft-Aperiodic Task Scheduling in Dynamic Priority Systems", submit to publication

[LEE] J. Lee, S. Lee and H. Kim, "Scheduling of Hard Aperiodic Tasks in Hybrid Static/Dynamic Priority Systems", ACM SIGPLAN Notices, Vol.30, No.11, November 1995, pp. 7-19.

[LEHOCZKY 92] J.P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems", Proceedings of the IEEE Real-Time Systems Symposium, pp. 110-123, December 1992.

[[LEHOCZKY 95] J.P. Lehoczky and S. R. Thuel, Scheduling Periodic and Aperiodic Tasks using the Slack Stealing Algorithm (Chapter 8), Advances in Real-Time Systems, (ed. S. Son) Prentice-Hall, Englewood Cliffs, NJ, 1995.

[LIU] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environments", Journal of the Association for Computing Machinery, Vol. 20, No.1, pp. 46-61, January 1973.

[SCHWAN] K. Schwan and H. Zhou, "Dynamic Scheduling of Had Real-Time Tasks and Real-Time Threads", IEEE Transactions on Software Engineering, Vol. 18, No. 8, August, 1992, pp.736-748.

[SHIN] K.G. Shin and Y.-C. Chang, "A Reservation-Based Algorithm for Scheduling Both Periodic and Aperiodic Real-Time Tasks", IEEE Transactions on Computers, Vol. 44, No.12, pp. 1405-1419, December, 1995.

[SPRUNT 88] B. Sprunt, J.P. Lehoczky, and L. Sha, "Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm", Proceedings of the IEEE Real-Time System Symposium, pp. 251-258, December, 1988.

[TIA] T.S. Tia, Utilizing Slack Time for Aperiodic and Sporadic Requests Scheduling in Real-Time Systems, Technical Report No. UIUCDCS-R-95-1906, University of Illinois, April, 1995.

Professor Heonshik Shin
Dept. of Computer Engineering,
College of Engineering, Seoul National University
Seoul, 151-742, Korea
Tel: +82-2-880-7295, Fax: +82-2-886-7589, shinhs@ce2.snu.ac.kr

June 6, 1996

Dear Professor Shin;

We are herewith submitting 5 copies of the paper entitled "***Scheduling Hard-Aperiodic Requests in Dynamic Priority Systems***" for the 3rd International Workshop on Real-Time Computing Systems and Applications, Oct. 30-Nov. 1, 1996, Seoul, Korea. Please be advised the following information:

1. Author Name, Address, E-mail, and Phone Number:

Hyungill Kim and Sungyoung Lee ,
Dept. of Computer Engineering Kyung Hee University,
Yongin-kun, Kyungki-do, 449-701, Korea
E-mail: hikim@oslab.kyunghee.ac.kr
slee@nms.kyunghee.ac.kr
Tel: +82-331-280-2514, Fax: +82-331-281-4965

Jongwon Lee,
Software Research Laboratories, Korea Telecom,
17, Woomyeon-dong, Seocho-gu, Seoul, 137-140, Korea
E-mail: jwlee@coral.kotel.co.kr
Tel: +822-526-6561, Fax: +822-526-5909

2. Abstract

In this paper, we present a hard-aperiodic task scheduling algorithm in dynamic priority systems. The proposed algorithm has an elegant acceptance test method for hard-aperiodic requests. The proposed scheduling scheme is based on the APS (Alternatively Priority Scheduling) algorithm which has a simple slack calculation mechanism. The operation of the algorithm is to reference the off-line built CTI (Critical Task Indicating) table, which is created by the deadlinewise preassignment policy, and choose either an EDF or a CEF (Critical Execution time First) algorithm alternatively at on-line. Since the APS algorithm adopts an optimistic slack calculation philosophy, it resolves the drawback of the slack-stealing based schedulings such as a high computational complexity to calculate the slack which in consequence makes them not be practical.

3. Keywords

Real-time scheduling, dynamic priority systems, deadline guarantees, hard-aperiodic task scheduling, acceptance test mechanism

4. Contact Point: Sungyoung Lee

Thank you very much for your concern to this matter.

Sincerely yours,

Sungyoung Lee,
Dept. of Computer Engineering,
Kyung Hee University,
Yongin-kun, Kyungki-do, 449-701, Korea