An Optimal Algorithm for Jointly Scheduling Sporadic and Periodic Tasks

Houssine Chetto

Institut Universitaire de Technologie de Nantes
Département d'Organisation et Gestion de la Production
La Chantrerie CP 3003
44087 Nantes Cedex 03
FRANCE

Abstract: In this paper, we describe an optimal approach for jointly scheduling periodic critical tasks and sporadic tasks. The problem is to schedule the tasks so that all the hard deadlines of periodic tasks are met and the response times for sporadic tasks are minimized. We show how optimal responsiveness can be attained by exploiting specific properties of the Earliest Deadline scheduling algorithm, in particular, how every occurring soft sporadic task is changing to a critical one by the adequate computation of a fictive deadline. Finally, to illustrate the performance of our approach, we provide a comparative study with other approaches and we report on simulation results.

Keywords: Dynamic Scheduling, Earliest Deadline, Periodic Critical Tasks, Real-time systems, Sporadic Tasks, Response Time.

Houssine Chetto was born in Berkane, Morrocco, in December 1952. He received the Maitrise d' Electronique et d'Automatique from the University of Orléans, Francein 1978, the degree of Docteur de 3ème Cycle in Concurrent Computing, in October 1981 and the degree of Docteur d'Etat in Scheduling in hard-real time systems, in December 1990.
Between 1981 and 1985 he was an Assistant Professor in the Department of Physics of the University of Fes, Morrocco. From 1985 to September 1988 he was employed by Ecole National Supérieure de Mecanique de Nantes, France, as an Assistant Professor at the Department of Automatic Control. Since 1988 he has been working as Assistant Professor at the Department of Production Management, Institut Universitaire de Technologie de Nantes. His fields of interest are concurrent computing, fault-tolerance and scheduling.

1. Introduction

For many computer applications such as automated manufacturing systems, nuclear power plants, robot arm control, etc., programs (generally called tasks) must satisfy specific timing constraints which are imposed by the physical process being controlled in order to avoid catastrophic results. We call such tasks which are required to respond to external and internal stimuli within a specified deadline, *critical tasks*; and when all the tasks are critical, the application is said to be *hard real-time*. In this paper, the critical tasks of interest run on a single processor machine that may be any node of a distributed system. This machine has its own private memory which is supplied with an operating system and the application software. It works in the feedback loop of the controlled system. Its goal is to derive inputs from sensors and then, its outputs are sent to control actuators or to update displays. To perform this control function which is initially well-defined, the computer has to run a set of tasks, cyclically, in an indefinite loop. Values of task periods then depend on the dynamics of the physical process associated with them. To ensure a correct execution of these tasks (i.e adherence with all their timing requirements), it appears that an efficient use of the processor by a careful scheduling is necessary.

Furthermore, the computer system has to cope with unpredictable changes in the environment whose effects can result in a sudden and temporary increase in processor workload. This means that there exist additional tasks which are non-periodic and lie dormant until they are activated. These so-called *sporadic tasks* represent application services such as maintenance, bookkeeping and alarm processing but also result from system activities such as bidding in a distributed system [1][2]. Although they are not necessarily critical, they can affect at a certain point in time the scheduling of critical periodic tasks since they share the processor with them. We call such a system in which only a subset of tasks is critical, a *semi-hard real-time system*.

An important topic in real-time systems research is the design of scheduling strategies with a view at meeting all the timing constraints of critical tasks while optimizing a performance criterion for soft tasks. Our goal here is to present a new approach that provides an optimal solution to this scheduling problem. The work described here is an extension of our own work that was initially reported in [3] on scheduling critical periodic tasks in presence of critical sporadic tasks. Clearly, this paper focuses on scheduling in a dynamic system where no a priori information on the arrival time or execution time of sporadic tasks, is known. We assume that sporadic tasks have the same priority and consequently, the arrival time will be used to break the competition tie on First Come First Serve (FCFS) basis. We will be precisely concerned with the problem of jointly scheduling the periodic and sporadic tasks so that the deadlines of periodic tasks are met and the response times for sporadic tasks are minimized. Our strategy is built upon the preemptive Earliest Deadline algorithm [4] which features ease of implementation and the best performance among the scheduling algorithms for critical tasks. We will show that our approach enables us to determine the response time of every sporadic task, as soon as it arrives.

The .paper is organized as follows: Section 2 provides some background material about dynamic real-time scheduling. In Section 3, we recall basic results, instrumental in the resolution of the scheduling problem. Section 4 describes the theoretical foundation of the proposed approach and provides an illustrative example. An outline of the scheduler is given in Section 5. This is followed by a comparative study involving Background, Polling and Deferral approaches in Section 6. Section 7 summarizes the significant features of our approach to dynamic task scheduling.

2.Problem Description

2.1 Scheduling Periodic Tasks

A periodic task set can be denoted as follows: $\mathcal{T} = \{\mathcal{T}_i\ (C_i,\ R_i,\ P_i),\ i=1 \text{ to } n\}$. In this characterization, every task $\mathcal{T}_i$ makes its initial request at time zero and its subsequent requests at time kP_i, $k=1,2,...$ The execution time needed by each request of $\mathcal{T}_i$ is C_i time units and a deadline for $\mathcal{T}_i$ occurs R_i units after each request by which $\mathcal{T}_i$ must have completed its execution. Throughout our discussion, we assume that a preemptive scheduling discipline is employed. Thus, a request for C_i units of execution time can be satisfied by one or more quanta which sum to C_i.

Many researchers have developed efficient scheduling algorithms specifically for periodic tasks. Liu and Layland [4] developed the Rate Monotonic (RM) algorithm which is a static priority driven algorithm. Under RM, higher priorities are assigned to tasks with shorter periods. They showed that this scheme was optimal among fixed priority schemes. However, they assumed that the relative deadline of a periodic task was equal to the period of the task. Under this hypothesis, they proved that RM had a worst case scheduling bound of ln2. That is, this algorithm guarantees that n periodic tasks can always be scheduled to meet deadlines if the processor utilization is less than $n(2^{1/n} - 1)$ which converges to 0.69 for large n. We recall that the processor utilization of the task set $\mathcal{T}$ is given by $\sum_{i=1}^{n} C_i / P_i$ and will be denoted by U.This scheduling bound can be increased by using a dynamic algorithm such as the Earliest Deadline (ED) algorithm. ED schedules at each instant of time t, the ready request (i.e. the request that may be processed and has not been completed yet) whose deadline is closest to t. It was proved to be the optimum preemptive scheduling algorithm [5][6]. That is, it produces a valid schedule (i.e. a schedule in which all the deadlines are met) for every schedulable task set. Furthermore, condition $U \leq 1$ (1) is necessary and sufficient to guarantee a valid schedule for $\mathcal{T}$ if the relative deadline R_i of every task $\mathcal{T}_i$ is equal to its period P_i. Else, condition $CH \leq 1$ (2) where $CH = \sum_{i=1}^{n} C_i / R_i$ has been proved to be sufficient. In all these studies, it is assumed that there is no time loss in preemption which means that the overhead originating from the task preemption (i.e. context switching) must not account for in the performance evaluation. The fundamental property of the schedule produced on $\mathcal{T}$ by any preemptive scheduling algorithm and in particular ED, is its cyclicity [7]. Let P= LCM (P_1, P_2,..., P_n) the base period be equal to the least common multiple of the periods P_1, P_2, ... P_n. This property means that the processor does exactly the same thing at time t>0 as at time t+kP (k=1, 2,...). So, studying the form of the schedule produced over an infinite length by ED amounts to studying the form of the schedule on the intervals [kP, (k+1)P], k=1, 2 ..., each of them being called a window.

2.2 Scheduling Sporadic Tasks

Typically, a sporadic task is a non-periodic task that occurs and requires to be run just once, dynamically. Consequently, sporadic tasks scheduling requires a dynamic approach which permits to determine schedules for tasks on the fly and can easily adapt to an unpredictable change in the processor workload. The problem of scheduling periodic tasks together with critical sporadic tasks has been extensively studied. In such a context, we say that a new occurring task is accepted if the scheduling algorithm can find a schedule for all the periodic tasks and the previously accepted sporadic tasks such that each task finishes by its deadline. Consequently, a scheduling algorithm is said to be optimal if any occurring task is accepted whenever possible. Mok [8] showed that ED was optimal for scheduling preemptable non-periodic tasks with arbitrary arrival times and hard deadlines. Mok also showed that the Least Slack time algorithm was optimal as well. The slack time is the time remaining until the task deadline reduces by the remaining task execution time. Ramamritham and Stankovic [1] described an acceptance test which was based on the non-preemptive Earliest Deadline algorithm and consequently provided a sub-optimal solution to the scheduling problem. An optimal solution was given in [3] which was based upon ED. Its extension as to taking into account precedence constraints is described in [9]. Schwan and Zhou [10] developed a dynamic scheduling algorithm for hard periodic and sporadic tasks, based upon ED. Its main advantage lies in its efficient implementation since its worst case complexity is O(nlogn).

Two basic approaches for processing soft sporadic tasks together with periodic tasks are Polling and Background. Polling consists in initially creating a periodic task for processing sporadic tasks. At regular intervals, the polling task (also called the server task) is requested to process the ready sporadic tasks. However, if no sporadic tasks are ready to be processed, the server task suspends itself until its next period and the time originally allocated for sporadic tasks are used instead by periodic tasks. In such a context, a sporadic task which occurs just after the server task has got suspended, must wait for the next request of the server task. Under Background technique, sporadic tasks are processed whenever the processor is idle i.e. all the ready periodic tasks are completed. When using these two approaches, periodic tasks can be scheduled by any priority driven algorithm. Simulations of these algorithms were reported in [11] in order to compare their performance with other heuristics. Here, performance is measured in terms of average response time. We refer the response time of a sporadic task to be the time elapsed from its occurring time to its completion time.

New algorithms have been proposed in [12] which provide better performance than Polling and Background: Priority Exchange (PE) and Deferrable Server (DS). These algorithms are built upon RM for scheduling periodic tasks and assume that a high priority periodic task can be dedicated to servicing sporadic tasks. They are called bandwidth preserving algorithms because they provide a mechanism for preserving the processor bandwidth allocated for sporadic tasks if, upon becoming available, the bandwidth is not immediately needed. More recently sophisticated approaches on static priority and on -line computation processor idle time, have been proposed [13] [14][15][16].

3. Basic Results

3.1 Motivations for ED

The scheduler of next generation real-time systems must be designed so as to guarantee real-time constraints of critical tasks, to minimize response time of soft tasks and moreover to support fault-tolerance [17]. Indeed, such real-time systems are very complex because of being distributed across a network and being capable of exhibiting adaptive and highly dynamic behaviour. The problem of designing a scheduling algorithm is then much more complicated than in classical centralized systems. Schedulers for these systems often consist of two components, namely a local scheduler and a distributed scheduler. The local scheduler first decides on whether critical tasks which arrive at a node can be scheduled on that node, and second attempts to schedule soft tasks as fast as possible. A distributed scheduler dynamically decides on where in the network, a soft task should be transferred to so as to minimize its response time, or a critical task that cannot be scheduled at a node should be transferred so as to meet its timing requirements. When the periodic workload of a node is static i.e. it consists of a fixed set of tasks whose characteristics are well- known before the system starts operate, it is often possible to use a static priority driven scheduler (such as RM) since adherence to timing requirements only depends on timing parameters of tasks which are assigned to that node. However, let us recall that one primary measure of a scheduling algorithm is its associated processor utilization level below which the deadlines of all tasks can be met. It has been proved that, from this point of view, ED is far more efficient than RM. Consequently, it permits to deal with a larger class of applications and is more suitable for highly dynamic systems. Furthermore, although RM is efficient to implement, the priority assignment obtained by it is not optimal for the general task model studied in this paper. Static priority driven algorithms generally involve smaller overhead in their implementation than the dynamic ones do. But this is no longer true under the assumption that the processor workload is composed of a mixed set of periodic and sporadic tasks. In such a context, obtaining a good solution to the scheduling problem will require priority exchanges in large numbers and consequently will involve as large overhead as a dynamic priority driven algorithm. Moreover, when there are more distinct task periods than available priority levels, some tasks with different periods must be assigned the same priority level. This insufficiency of priority levels then leads to a reduction of processor scheduling potential. This phenomenon is particularly true for reconfigurable distributed systems in which a node can be transiently overloaded. Indeed, the failure of one or several nodes will provoke the migration of periodic tasks on the remaining safety nodes for which a static allocation of priority is no longer appropriate. Now, let us review briefly the fundamental properties of ED which were stated in [3] and [18] and are the foundation of our approach for scheduling a mixed set of periodic and sporadic tasks.

3.2 Off-line Computations

If a uniprocessor system solely supports periodic tasks, it is easy to know exactly what is done by the processor at any moment since the schedule which is to be executed can be determined at system initialization time and is not modified during the lifetime of the system. In particular, it is interesting to determine the localization and the duration of time intervals (called idle times) during which the processor is not busy executing a periodic task. Naturally, we assume that tasks are executed as soon as possible i.e. as soon as the processor is idle and all the tasks considered to be more urgent have been processed.

We can imagine an implementation of ED (denoted EDL) that amounts to executing tasks as late as possible. Indeed, it was proved in [3] that such implementation of ED led to maximizing processor idle time as soon as possible. Then the schedule so constructed can be described by:

- $\mathbf{K}=(k_0, k_1, \ldots k_i, k_{i+1}, \ldots k_q)$ with $k_i<k_{i+1}$, $k_0=0$ and $kq=P-\inf\{x_i; 1\leq i\leq n\}$ where $x_i=P_i-R_i$ for all $1\leq i\leq n$. $\mathbf{K}$ is called the *deadline vector* and is obtained from the distinct deadlines of requests within $[0, P]$.
- $\mathcal{D}=(\Delta_0, \Delta_1, \ldots, \Delta_i, \Delta_{i+1}, \ldots, \Delta_q)$. $\mathcal{D}$ is called the *idle time vector under EDL* and Δ_i denotes the length of the idle time that follows time k_i in the schedule produced by EDL for $\mathcal{T}$ within [0,P]. It was proved that Δ_i is given by the following recurrent formulae:

$$\Delta_q=\inf\{x_i, 1\leq i\leq n\} \tag{3}$$

$$\Delta_i = \sup\left(0, (P - k_i) - \sum_{j=1}^{n} \left\lceil \frac{P - x_j - k_i}{P_j} \right\rceil .\right.$$

$$\left. C_j - \sum_{k=i+1}^{q} \Delta_i \right) \; i = 0, 1, \ldots, q-1 \qquad (4).$$

where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x. From definition of $\mathcal{D}$, we immediately deduce that $\sum_{i=0}^{q} \Delta_i = \Phi$ where Φ denotes the quantity of processor idle time within any window and is given by P(1-U) (5).
Description of the schedule produced by EDL for $\mathcal{T}$ is then madein O(N) operations thanks to recurrent formulae (3) and (4). Here, N denotes the total number of requests within [0,P].

3.3 Dynamic Scheduling

Now, let us describe the dynamic workload imposed on the machine at any current time τ by $\mathcal{T}$. It is composed of the current requests which have occurred at or before τ and have not been completed at τ and future requests which have not started their execution at τ. The set of all these requests (denoted by $\mathcal{T}(\tau)$) can be assimilated to a set of non-periodic tasks, the dynamic parameters of which are deduced from those of periodic tasks. In what follows, it will be denoted by $C_i(\tau)$ and d_i the dynamic execution time and current deadline of task T_i at time τ. In [3], it has been stated that applying EDL to $\mathcal{T}(\tau)$ results in the maximization of total idle time within any interval $[\tau,t]$, $\tau \le t$ while maintaining adherence with all the timing requirements. This fundamental property is formalized in the following theorem:

Theorem 1: (For the proof, see [3])
For any scheduling algorithm X and any instant t such that $t \ge \tau$,

$$\Omega^{X}_{\mathcal{T}(\tau)}(\tau,t) \le \Omega^{EDL}_{\mathcal{T}(\tau)}(\tau,t) \qquad (6)$$

where $\Omega^{X}_{\mathcal{T}(\tau)}(\tau,t)$ denotes the quantity of processor idle time between τ and t for the schedule produced by algorithm X on task set $\mathcal{T}(\tau)$. From this theorem, we may conclude that applying EDL to any set of hard deadline tasks will result in maximizing total idle time within any time interval $[\tau,t]$, $0 \le t \le P$.
We will show in the next section that, in allowing for the arrival of soft tasks and in scheduling them so as to minimize response times, we need follow a similar approach. In order to compute the quantity $\Omega^{EDL}_{\mathcal{T}(\tau)}(\tau,d)$, we need construct the schedule produced by EDL on the task set $\mathcal{T}(\tau)$ from the current time τ. It was proved that such a schedule can be described by a *dynamic deadline vector* and a *dynamic idle time vector* as follows:
- $\mathcal{K}(\tau) = (k_h, k_{h+1}, \ldots, k_q)$ with $k_h = \tau$ and $k_{h+1} = \min\{k_i \in \mathcal{K};\ k_i > \tau\}$. $\mathcal{K}(\tau)$ is obtained from the distinct deadlines of requests posterior to τ in the current window plus the time instant τ. We assume that all these dynamic parameters are measured from the beginning of the window which can be considered as a new time zero.
- $\mathcal{D}(\tau) = (\Delta_h(\tau), \Delta_{h+1}(\tau), \ldots, \Delta_q(\tau))$. $\Delta_i(\tau)$ denotes the length of the idle time that follows time k_i in the schedule produced by EDL for $\mathcal{T}(\tau)$. Let $M = \max\{d_j;\ \mathcal{T}_j \in \mathcal{T}(\tau)\}$. M denotes the latest deadline among those of current requests of periodic tasks. Let index l be such that $k_l = \min\{k_i / k_i \ge M\}$. Finally, it was proved that:

$$\Delta_i(\tau) = \Delta_i \text{ for } i = l \text{ to } q, \qquad (7)$$

$$\Delta_i(\tau) = \sup\Bigg(0, (P - k_i) - \sum_{\substack{j=1 \\ d_j > k_i}}^{n} C_j(\tau) - \sum_{j=1}^{n} \left\lceil \frac{P - x_j - \sup(k_i, d_j)}{P_j} \right\rceil \cdot C_j - \sum_{k=i+1}^{q} \Delta_k(\tau)\Bigg) \text{ for } i = h \text{ to } l-1 \qquad (8)$$

3.4 Illustration

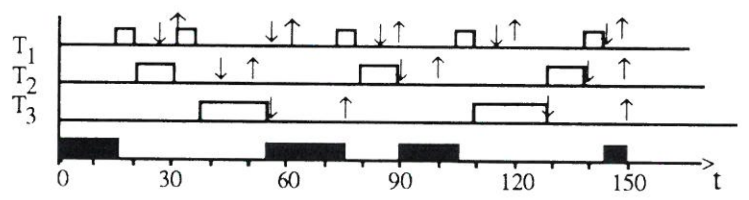
Let $\mathcal{T} = \{T_1(5, 25, 30), T_2(10, 40, 50), T_3(20, 55, 75)\}$. $\mathcal{T}$ is a schedulable periodic task set since CH = 0.813 and verifies (2). From construction of $\mathcal{K}$ and thanks to formulae (3) and (4), we deduce the length of every idle time within [0, P] where P = 150, for the schedules produced by EDL (see Table 1), respectively.
Results described in Table 1 are easily verifiable by constructing the schedules produced by EDL (see Figure 1). The corresponding schedule produced by ED is reported in Figure 2.

i	0	1	2	3	4	5	6	7	8	9
k_i	0	25	40	55	85	90	115	135	140	145
Δ_i	15	0	0	20	0	15	0	0	0	5

Table 1. Values of $\mathcal{K}$ and $\mathcal{D}$

i	4	5	6	7	8	9
k_i	85	90	115	135	140	145
$\Delta_i(\tau)$	5	20	5	0	0	5

Table 2. Values of $\mathcal{K}(\tau)$ and $\mathcal{D}(\tau)$



■ :Processor idle time ↑ :request ↓ :deadline

Figure 1. Schedule Produced by EDL on $\mathcal{T}$

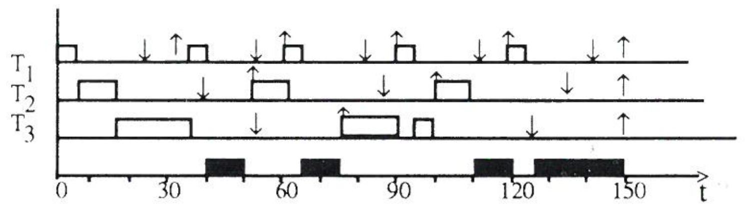


Figure 2. Schedule Produced by ED on $\mathcal{T}$

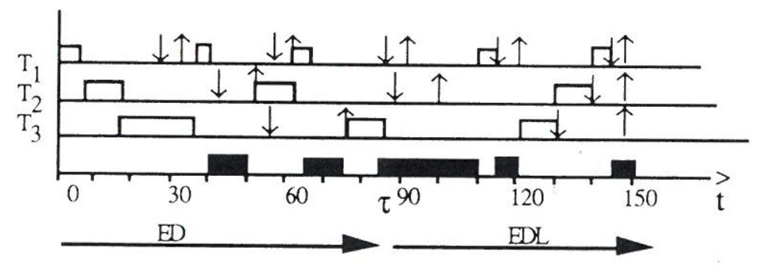


Figure 3. Schedule Produced by EDL at time τ=85

Now, let us consider the current time instant τ=85 and assume that tasks in $\mathcal{T}$ are scheduled by ED from 0 to 85. From construction of $K(\tau)$ and thanks to formulae (7) and (8), we deduce the length of every idle time within the current window for the schedule produced by EDL on

τ=85the task set $\mathcal{T}(\tau)$ (see Table 2). Chronogram of Figure 3 enables us to verify values of Table 2 and provides an illustration of Theorem 1.

4. Dynamic Scheduling of Sporadic and Periodic Tasks

4.1 Problem Statement

In this section, we will be concerned with the problem of finding an optimal scheduling algorithm for a mixed set of critical periodic tasks and soft sporadic tasks. By optimality, we mean that the algorithm produces a schedule in which all the sporadic tasks are executed as fast as possible and deadlines of periodic tasks are met. As mentioned in Section 1, sporadic tasks are assumed to have the same importance relating to the application and consequently they are served in FCFS order. Our objective then amounts to determining how to schedule sporadic tasks with respect to periodic tasks.
As ED is optimal for scheduling periodic and non-periodic tasks with hard deadlines, the basic idea of our approach is to change a set of soft sporadic tasks to a set of critical sporadic tasks. Then, we will deal with the problem of finding, for each occurring sporadic task, the minimal deadline that can be associated with it so that this task is executed with a minimal response time. This work method presents the following advantages:

- A unique scheduler employing the Earliest Deadline strategy may be used to schedule sporadic tasks and periodic tasks without discrimination.
- Minimization of the response time of any sporadic task is obtained due to the computation of an adequate fictive deadline.

So, we present an algorithm consisting of two basic parts. In the first part, fictive deadlines are computed for the sporadic tasks and, in the second part, they are used to construct a priority list where requests for periodic tasks and sporadic tasks are ordered by decreasing deadlines.

4.2 Theoretical Foundation of the Approach

Let τ be the current time which coincides with the arrival time of a soft sporadic task **R**. Upon its arrival, **R** is characterized by its execution time C. First, let us assume that there are no other sporadic tasks available on the machine at time τ. Let us prove that the problem of determining deadline d of task **R** amounts to the problem of determining the time instant t ($t \geq \tau$) that verifies $\Omega^{EDL}_{\mathcal{T}(\tau)}(\tau, t) = C$.

Theorem 2:
By assigning a fictive deadline d to **R** *that verifies* $\Omega^{EDL}_{\mathcal{T}(\tau)}(\tau, d) = C$ (9) . **R** *is executed within a minimum response time while ensuring deadlines of periodic tasks*

Proof: From Theorem 1, $\Omega^{EDL}_{\mathcal{T}(\tau)}(\tau, d)$ represents the maximum processor idle time that can be recovered to process additional tasks within [τ, d]. Since this quantity is equal to the execution time of task **R**, we are sure that **R** cannot finish its execution before time d. If we denote by f the completion time of **R**, we have $f \geq d$ (10). Value ($d-\tau$) then provides a lower bound on the response time of task **R**. Moreover, we know that **R** is feasibly scheduled when assigned deadline d. This means that $f \leq d$ (11). From (10) and (11), we deduce that f=d and consequently **R** will be executed with a minimal response time equal to ($d-\tau$).□
By Theorem 2, equality (9) enables us to guarantee that task R with a fictive deadline d,is processed as soon as possible while producing a valid schedule for periodic tasks.
Now, let us assume that other sporadic tasks are available on the machine at τ. This means that such tasks have occurred before τ and are not completed at τ. Let us denote by $\mathbf{T}(\tau)$ this sporadic task set and described as follows: $\mathbf{T}(\tau)$=

$\{R_i(C_i(\tau),d_i),\ i=1 \text{ to } m(\tau)\}$. In this characterization, task $\mathbf{R}_i$ is ready to be processed at τ. Value $C_i(\tau)$ represents the dynamic execution time of $\mathbf{R}_i$ at τ (i.e. its remaining execution time). d_i is the fictive deadline of $\mathbf{R}_i$ which has been associated with it upon its arrival. We assume that $\mathbf{T}(\tau)$ is ordered such that $i<j$ implies $d_i \leq d_j$.

$\mathbf{T}(\tau)$ can be considered as a set of critical sporadic tasks. Consequently, let us introduce for any task $\mathbf{R}_i$ and any instant t, the quantity $\delta_i(t)$ defined as follows: $\delta_i(t) = \Omega^{EDL}_{\mathcal{T}(\tau)}(\tau,d_i) - \sum_{j=1}^{i} C_j(t)$ (12). $\delta_i(t)$ is called laxity of $\mathbf{R}_i$ at t. It represents the surplus processing power within the scheduling interval of $\mathbf{R}_i$ such that all periodic and sporadic tasks with deadline less than or equal to d_i are feasibly scheduled. For the purpose of deducing the fictive deadline of a new occurring task, the following propositions are of a more practical interest:

Proposition 3:

Let $\boldsymbol{R}_i$ and $\boldsymbol{R}_j$ be two sporadic tasks which respectively occur at times τ_i and τ_j such that $\tau_i<\tau_j$. Then, $\boldsymbol{R}_i$ and $\boldsymbol{R}_j$ must be assigned fictive deadlines d_i and d_j, respectively, which verify $d_i<d_j$.

Proof: The proof is immediate since we assume that sporadic tasks are served on FCFS basis. $\tau_i<\tau_j$ then implies that $\mathbf{R}_i$ must be served before $\mathbf{R}_j$. Since ED is used to schedule the tasks by decreasing deadline, this means that $d_i<d_j$.□

From **Proposition 3**, we may immediately deduce the two following results: there are no preemptions between sporadic tasks, and the set of sporadic tasks $\mathbf{T}(\tau)$ can be viewed as a unique critical sporadic task, the dynamic execution time of which is equal to the sum of the dynamic execution times of the tasks in $\mathbf{T}(\tau)$, the deadline of which is equal to the deadline of the task which was the last to arrive before τ.

Proposition 4:

For any sporadic task $\boldsymbol{R}_i$ which is ready at time t and not completed at time $t+\theta$ with $\theta \geq 0$, we have $\delta_i(t+\theta) = \delta_i(t)$. (13)

Proof: First, let us assume that no sporadic task occurs within the time interval $[t, t+\theta]$. By definition, $\delta_i(t) = \Omega^{EDL}_{\mathcal{T}(t)}(t,d_i) - \sum_{j=1}^{i} C_j(t)$ and

$$\delta_i(t+\theta) = \Omega^{EDL}_{\mathcal{T}(t+\theta)}(t+\theta,d_i) - \sum_{j=1}^{i} C_j(t+\theta)$$

Let $x = \Omega^{EDL}_{\mathcal{T}(t)}(t,d_i) - \Omega^{EDL}_{\mathcal{T}(t+\theta)}(t+\theta,d_i)$. The maximum available time for processing sporadic tasks within $[t, d_i]$ is given by $\Omega^{EDL}_{\mathcal{T}(t)}(t,d_i)$. At time $t+\theta$, the dynamic workload inflicted by periodic tasks and resulting from the processor activity between t and $t+\theta$, is represented by the task set $\mathcal{T}(t+\theta)$. So, $\Omega^{EDL}_{\mathcal{T}(t+\theta)}(t+\theta,d_i)$ gives the total idle time within $[t+\theta, d_i]$ by scheduling according to EDL all the requests of $\mathbf{T}(t)$ which are unfulfilled at time $t+\theta$. So, x represents the total processor time within $[t, t+\theta]$ which has not been dedicated to periodic tasks. Since all sporadic tasks are available from t, task $\mathbf{R}_i$ is not completed at $t+\theta$ and tasks are scheduled according to EDS, it comes that x represents the total processor time dedicated to sporadic tasks with deadline less than or equal to d_i within $[t,t+\theta]$.

Let $y = \sum_{j=1}^{i} C_j(t) - \sum_{j=1}^{i} C_j(t+\theta)$. It is clear that x=y and consequently equality (13) is verified.

Now, let us assume that task $\mathbf{R}_j$ occurs at time $t+\theta$ and no task occurred between t and $t+\theta$. From what precedes, we have $\delta_i(t+\theta-\varepsilon) = \delta_i(t)$ (16) where $\varepsilon \to 0$. From proposition 3, we have $d_i<d_j$ for all i $\{1...m(t)\}$. Consequently, from definition of the laxity, the occurrence of $\mathbf{R}_j$ at time $t+\theta$ does not modify the laxity of $\mathbf{R}_i$ computed at time $t+\theta-\varepsilon$ where $\varepsilon \to 0$. From (14), it comes that $\delta_i(t+\theta) = \delta_i(t)$ for all i $\{1...m(t)\}$.□

Proposition 5:

Every task $\boldsymbol{R}_i$ of $\boldsymbol{T}(\tau)$ is feasibly executed if $\delta_i(\tau) \geq 0$ (15).

Proof: Let us assume that condition (15) holds for any task $\mathbf{R}_i$ of $\mathbf{T}(\tau)$. From Theorem 1, applying EDL to $\mathcal{T}(\tau)$ from time τ enables us to guarantee a feasible execution of any request of periodic task while maximizing the quantity of processor idle time which could be recovered within the time interval $[\tau, d_i]$. This quantity is given by $\Omega^{EDL}_{\mathcal{T}(\tau)}(\tau,d_i)$ and assumed to be greater than or equal to $\sum_{j=1}^{i} C_j(\tau)$. As this quantity represents the total computation time required by all the tasks with a deadline less than or equal to d_i and as sporadic tasks are scheduled by decreasing deadline, we conclude that there exists at least one schedule in which task $\mathbf{R}_i$ is completed before its deadline d_i. Such a schedule can be obtained by applying the optimal algorithm ED to $\mathbf{T}(\tau)\mathrm{U}\mathcal{T}(\tau)$.□

We are now prepared to state our main result:

Theorem 6:

Every task $\mathbf{R}_i$ of $\mathbf{T}(\tau)$ is executed with a minimal response time if it is assigned fictive deadline so that $\delta_i(\tau)=0$ (16).

Proof:

Let us consider task $\mathbf{R}_i$ at time τ such that $\delta_i(\tau)=0$ and prove that $\mathbf{R}_i$ will be executed in a minimal response time.

From proposition 5, $\delta_i(\tau)=0$ implies that $\mathbf{R}_i$ is feasibly scheduled, which means that $f_i \leq d_i$ (17). Furthermore, as in the previous proof, $\sum_{j=1}^{i} C_j(\tau) = \Omega^{EDL}_{\mathcal{T}(\tau)}(\tau,d_i)$ implies that $\mathbf{R}_i$ cannot terminate before d_i i.e. $f_i \geq d_i$ (18). From (17) and (18), it comes that the response time of $\mathbf{R}_i$ is equal to $d_i-\tau$ and is minimal.□

Theorem 6 enables us to state the optimality of the proposed approach: it guarantees that any occurring task will be executed in a minimal response time if this task is transformed into a critical one, the deadline of which must render its laxity equal to zero. Moreover, from proposition 4, we know that the laxity of any sporadic task remains constant for all the lifetime of the task. Consequently, if the laxity of any task is equal to zero upon its arrival, this laxity will remain equal to zero up to its completion date.

4.3 Computation of Fictive Deadlines

Now, let us derive the value of d defined as the deadline of the new occurring task $\mathbf{R}$. For this purpose, we will have to make use of $I(\tau)$ and $A(\tau)$ respectively, defined as the remaining idle time and the total computation time required by sporadic tasks, from the beginning of the current window up to τ. We note that the remaining idle time from time τ up to the end of this window is given by $\Phi - A(\tau) - I(\tau)$ and denoted by $\Phi(\tau)$. Let $C(\tau) = \sum_{R_j \in \mathbf{T}(\tau) \cup \{R\}} C_j(\tau)$ and denote the total execution time required by sporadic tasks from time τ. We need distinguish between the two following cases:

–Case 1: $\Phi(\tau) > C(\tau)$

This means that deadline d occurs in the current window. Then, we have to determine the earliest deadline k which occurs in $[\tau, \lceil \frac{\tau}{P} \rceil P]$ that verifies $\Omega^{EDL}_{\mathcal{T}(\tau)}(\tau,k) \geq C(\tau)$. This amounts to computing the smallest index j in $\mathcal{D}(\tau)$ that verifies $\sum_{i=h}^{j} \Delta_i(\tau) \geq C(\tau)$. It follows that

$$d = k_j + \left(\sum_{i=h}^{j} \Delta_i(\tau) - C(\tau) \right) \quad (19).$$

–Case 2: Else

We need determine the most imminent window that contains time d and the earliest deadline in that window which is greater than or equal to d. This amounts to computing first, the smallest integer x that verifies $\Phi(\tau) + x.\Phi \geq C(\tau)$ and then, the smallest index j in $\mathcal{D}$ that verifies $\Phi(\tau) + x.\Phi + \sum_{i=h}^{j} \Delta_i \geq C(\tau)$. x will represent the number of successive windows between the current window and the window in which d occurs. Time k_j represents the deadline of a periodic request in that window which is followed by an idle time interval containing the deadline d. It comes that

$$d = (x+1).P + k_j + (\Phi(\tau) + x.\Phi + \sum_{i=h}^{j} \Delta_i - C(\tau)) \, (20).$$

In case 1, deadline d is computed from the dynamic idle time vector $\mathcal{D}(\tau)$ and in case 2 from the static idle time vector $\mathcal{D}$ which is assumed to be available from the initialization time.

4.4 Example

Let $\mathcal{T}$ be the task set shown in the example of Section 3. First, let us assume that $\mathbf{R}_1$ arrives at time τ_1 =85 with execution time C_1=25. We note that Φ= 55 and $\Phi(\tau_1)$= 35. Values of $\mathbf{K}(\tau_1)$ and

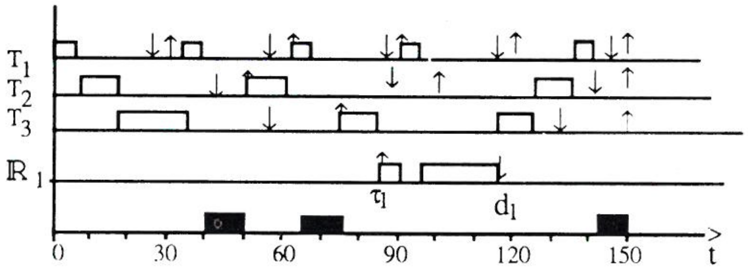


Figure 4. Schedule Produced on $\mathcal{T} \cup \{\mathbf{R}_1\}$

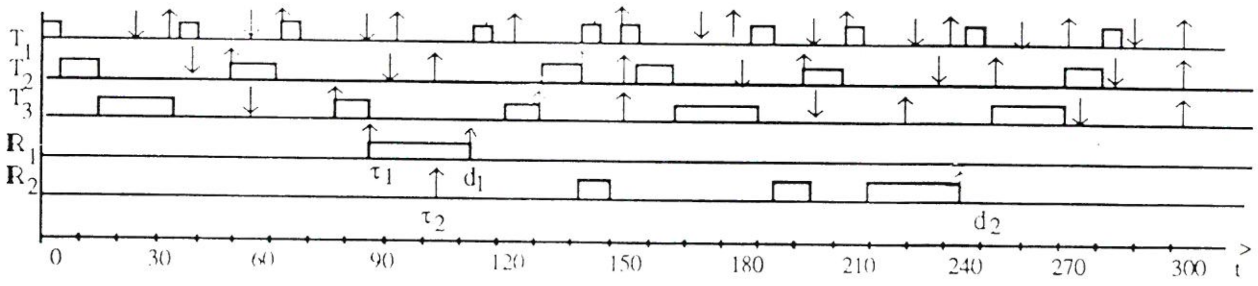


Figure 5. Schedule Produced on $\mathcal{T} \cup \{\mathbf{R}_1, \mathbf{R}_2\}$

$\mathcal{D}(\tau_1)$ are given in Figure 4 and Figure 5, respectively. Since $\Phi(\tau_1) \geq C_1$, we know that d_1 belongs to the current window and consequently is less than or equal to time 150. The smallest index j in $\mathcal{D}(\tau_1)$ that verifies $\sum_{i=h}^{j} \Delta_i(\tau_1) \geq C_1$ corresponds to k_j=90. Thanks to formula (20), we obtain d_1=110. Figure 4 displays the effective schedule which would be produced within the entire window [0,150]. Both periodic tasks and $\mathbf{R}_1$ with a fictive deadline d_1 are jointly scheduled according to EDS.

Now let us consider the occurrence of $\mathbf{R}_2$ at time τ_2=100 with execution time C_2= 50.We have $A(\tau_2)$=10, $I(\tau_2)$=20 , $C_1(\tau_2)$=18. It follows that $C(\tau_2)$= 68 and $\Phi(\tau_2)$=25. Since $\Phi(\tau_2)<C(\tau_2)$, we have to determine the smallest integer x and the smallest index j in $\mathcal{D}$ that verifies $\Phi(\tau_2) + x.\Phi + \sum_{i=h}^{j} \Delta_i \geq C(\tau_2)$. Finally, we obtain x=0 and k_j=90. It follows that d_2=248. Figure 5 displays the schedule effectively produced by EDS on the task set $\mathcal{T} \cup \{R_1,R_2\}$ within [0,300]. On this example, we can make the following remarks: there is no idle time from the arrival of a sporadic task until its completion time, the number of preemptions of sporadic tasks is minimized and all deadlines of periodic tasks are met. This enables us to state that response times of sporadic tasks are minimized, and shows the optimality of the strategy for this example and the minimal overhead involved by its implementation.

5. Implementation Considerations

The procedure that implements the acceptance of a new occurring task and its insertion at the scheduler level will need use several data structures. The static deadline vector $\mathcal{K}$ and the static idle time vector $\mathcal{D}$ are maintained in the Static Deadline Table (SDT) and the Static Idle Time Table (SIT) respectively. The dynamic deadline vector $\mathcal{K}(\tau)$ and the dynamic idle time vector $\mathcal{D}(\tau)$ are maintained in the Dynamic Deadline Table (DDT) and the Dynamic Idle time Table (DIT) respectively. Information on sporadic tasks such as fictive deadlines and dynamic execution times, is maintained in a data structure called the Sporadic Task Table (STT). During the operation of the system, the procedure uses the STT in conjunction with the Periodic Task Table (PTT). Each entry in the PTT contains a period, a dynamic execution time and a deadline. Tasks in PTT and STT are ordered by their deadlines. Dynamic values $A(\tau)$ and $I(\tau)$ which are re-initialized to zero at the beginning of every window, must be available for the procedure described in Figure 6.

Clearly, the algorithm described in Figure 6 runs in 0 (N) time in the worst case where N represents the number of distinct requests within a window. In the best case, the routine only needs the static idle time vector which has been computed at system initialization and consequently runs in 0(1) time. This makes the proposed approach be an efficient one in the sense that overhead induced whenever a sporadic task arrives, is low.

6. Experiments and Results

6.1 Simulation Model

To reiterate, the purpose of these studies was to bring to light the advantages of our approach over the techniques that have already been proposed. Clearly, this asks for generating representative task sets and then for evaluating each strategy with respect to the task sets. For this experiment, we have developed a task set generator. The task generation procedure has been parameterized so that it produces task sets having different degrees of scheduling difficulty and consequently allows us to provide an objective evaluation of our strategy. The generator is used to produce a task set from the following input parameters:

- the number of periodic tasks (n),
- the least common multiple of periods (P),
- the periodic task load (U_p),
- the average execution time of sporadic tasks ($C_{aver.}$),
- the average inter-arrival interval of sporadic tasks ($I_{aver.}$), and
- the duration of simulation.

The output parameters of this generator are timing parameters of the tasks (i.e. period (P_i),

```
Let .P be the least common multiple of the periods
    .U be the utilization factor of periodic tasks
    .f=P(1-U)
    .t be the current clock time
    .C(t) be the total remaining execution time of sporadic tasks
    .A(t) be the time used by sporadic tasks in the current window up to t
    .I(t) be the remaining idle time in the current window up to t

begin
  .Insert task R with execution time C after the last entry of the STT
  .C(t):=C(t)+C
  .f(t):=f- A(t)- I(t)
  .if f(t)≥C(t) then
     begin (* Case 1*)
        (* Determination of the Dynamic idle time vector*)
        find the last entry h in SDT such that $k_h \le t$
  initialize DDT with the h first entries of SDT
        for each entry i from the last one down to the first one
           of DDT do compute $D_i(t)$ thanks to () end do
        (*Computation of deadline d*)
        Som:=0
        for each entry i from the first one  of DIT
           repeat Som:=Som+ $D_i$ (t) until Som≥ C(t)
           d:= $k_j$ +(Som - C(t))
     end (* Case 1*)
     else
        begin (*Case 2*)
           (* Computation of the window which contains d*)
           x:=0; s:= f(t)
           repeat
              s:=s+f
              x:=x+1
           until s>C(t)
           (*Computation of deadline d*)
           Som:=s
           for each entry i from the first one  of SIT
              repeat Som:=Som+ $D_i$ until Som≥ C(t)
           d:= (x+1) P + $k_j$ +(Som - C(t))
        end (*Case 2*)
  .Assign deadline d to task R
end.
```

Figure 6. Description of the Scheduler

			Execution time							
i	P_i	R_i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
1	84	70	1	1	2	2	4	4	4	5
2	105	98	1	2	2	3	4	5	6	7
3	112	90	1	2	3	4	4	6	5	6
4	120	115	1	2	3	4	4	5	6	8
5	140	118	1	2	3	4	6	7	6	8
6	168	152	1	2	3	5	6	9	9	11
7	210	204	1	3	4	6	7	9	11	13
8	240	240	2	4	4	8	7	12	12	16
9	286	271	2	5	6	9	9	14	14	17
10	336	320	2	6	7	11	12	18	16	19
11	420	405	3	6	8	13	16	18	20	22
12	560	553	4	9	12	17	17	26	32	35
13	840	790	6	14	17	24	38	41	52	49
Utilization factor U_s			0.11	0.21	0.27	0.39	0.47	0.62	0.66	0.78

Table 3. Periodic Task Sets

# of Tasks	C_{nim}	C_{max}	C_{aver}	I_{min}	I_{max}	$I_{aver.}$	U_s
25	1	196	54	107	399	262	0.21

Table 4. Sporadic Task Set Characterization

	BG		PO		DS		EDL	
U_p	RT	% of Preem.	RT	% of Preem.	RT	% of Preem.	RT	% of Preem.
0.11	65	0.68	64	0.64	61	0.78	60	0.43
0.21	74	2.41	129	0.88	76	0.80	61	0.56
0.27	81	2.52	187	1.08	89	0.96	61	0.52
0.39	120	2.72	265	1.72	108	1.52	63	0.56
0.47	133	2.96	433	2.44	113	1.92	67	0.56
0.62	229	3.26	3394	7.72	206	3.64	86	0.72
0.66	258	3.52	5738	13.38	243	3.64	92	0.76
0.78	574	3.44	9834	53.68	562	5.32	180	1.12

Table 5. Response Time (RT) and Preemption Ratio (% of preem.)

critical delay (R_i), execution time (C_i) for periodic tasks, and the arrival time and execution time for sporadic tasks). In simulations reported here, parameters n and P take constant values of 13 and 3360 respectively. We have considered 8 periodic task sets (denoted S_i) for Up that uniformly varies from 0.11 to 0.78 and one set of 25 sporadic tasks with the utilization factor (denoted U_s, given by $C_{aver.}/I_{aver.}$) equal to 0.21. More precisely, for a given periodic task load U_p, C_i is chosen to be proportional to P_i with a minimal value equal to 1.
Parameters of periodic and sporadic tasks are respectively reported in Table 3 and Table 4. C_{min}, C_{max}, I_{min} and I_{max} respectively, denote the minimum and the maximum execution time, the minimum and the maximum inter-arrival interval of sporadic tasks.

6.2 Performance Comparison

The task sets resulting from different combinations of Up and Us have been successively scheduled according to the following strategies: BackGround (BG), Polling (PO), Deferral Server (DS) and EDL. Let remind of PO and DS as based on Deadline Monotonic which is an optimal static priority driven algorithm and schedules the tasks by the increasing order of their critical delay.
Two criteria have been applied in performance evaluation: the average response time (RT) and the preemption ratio (% of Preem.). This ratio is defined as the total number of preemptions of sporadic tasks per number of tasks. The preemption ratio provides us an indication about the timing overhead incurred by context-switching. Indeed, the efficiency of a scheduling algorithm depends on the complexity of this algorithm and on the number of involved preemptions.
Table 5 shows the average response time and the preemption ratio. Based on simulation results, the following remarks are to be made:

- As expected, EDL performs the best on all accounts and for any system load. The main reason for its better performance is that it dynamically recovers the maximum available processor idle time as soon as possible.
- As such an approach is based on the Earliest Deadline first algorithm, the number of preemptions is minimized.
- Under light load, the difference is not very significant. For systems with medium load, the performances of EDL strategy are significantly higher than those of other strategies. We can note that for U_p=0.47, the average response time provided by EDL is less than half that provided by any other strategy.
- When the system is heavily loaded, the performances of all strategies but EDL degrade and become unacceptably bad.

7. Conclusion

An approach was proposed in this paper to achieve an optimal scheduling in a semi-hard real-time system. Real-time software was referred to as a set of periodic tasks initially assigned to a monoprocessor machine, and in addition, as non-periodic tasks (said to be sporadic) occurring and requiring to be run on this machine at unpredictable times. The problem was to schedule the periodic and sporadic tasks so that hard deadlines of periodic tasks should be met and response times for sporadic tasks should be minimized.
One could imagine that dynamic priority driven scheduling algorithms are less suitable than static priority driven ones, whose implementation is simple and involves little overhead. However, the latter does not provide all the flexibility and predictability which are being increasingly required by next generation systems. These systems are highly dynamic and imply that the scheduler has been thus designed as to be able to cope with dynamic changes in processor workload.
In this paper we developed a dynamic preemptive scheduling strategy with a view at achieving a high flexibility. The crux of our approach lies in the ability of having a precise knowledge the maximum processor time which can be recovered and then dedicated to sporadic tasks. Optimal responsiveness to sporadic tasks is attained by computing fictive deadlines and then bny applying the famous Earliest Deadline

scheduling algorithm to both sporadic and periodic tasks. Such a method is efficient since it can be implemented in linear time and optimal since sporadic tasks are always processed as soon as possible while maintaining a feasible execution of periodic tasks.

REFERENCES

1. RAMAMRITHAM, K. and STANKOVIC, J.A., **Dynamic Task Scheduling in Distributed Hard Real-Time Systems,** IEEE SOFTWARE , Vol.1, 1984 .

2. ZHAO, W. and RAMAMRITHAM, K., **Distributed Scheduling Using Bidding and Focussed Addressing** , Proc. Real Time Systems, Symp. San Diego, CA,1985.

3. CHETTO, H. and CHETTO, M., **Some Results of the Earliest Deadline Scheduling Algorithm.** IEEE TRANS. ON SOFTWARE ENGINEERING 15, 10 October 1989.

4. LIU, C.L. and LAYLAND, J.W. , **Scheduling Algorithms for Multipro-gramming in a Hard Real-Time Environment,** JACM 20, 1 January 1973.

5. SERLIN, O., **Scheduling of Time Critical Processes** , Proc. of the Joint Computers Conf., 40, 1972.

6. LABETOULLE, J., **Some Theorems on Real Time Scheduling,** in E. Gelenbe and R. Mahl (Eds.) Computer Architectures and Networks, NORTH-HOLLAND, Amsterdam, 1974. .

7. LEUNG, J.Y.K. and MERRIL, M.L., **A Note on Preemptive Scheduling of Periodic Real Time Tasks,** INFORMATION PROCESSING LETTERS, 20, 3, 1980.

8. MOK, A.K., **Fundamental Design Pro-blems of Distributed Systems for the Hard Real-Time Environment,** Ph. D Thesis, M.I.T, 1983.

9. CHETTO, H., SILLY, M. and BOUCHENTOUF, T., **Dynamic Scheduling of Real-Time Tasks under Precedence Constraints,** THE JOURNAL OF REAL TIME SYSTEMS, 2, 1990.

10. SCHWAN, K. and ZHOU, H, **Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads,** IEEE TRANS. ON SOFTWARE ENGINEERING, 18 , 8, August 1992 .

11. LEHOCZKY, J.P., SHA, L. and STROSNIDER, J.K., **Enhanced Aperiodic Responsiveness in Hard Real-Time Environments,** Proc. of Real-Time Systems Symposium, IEEE, San Jose, CA, December 1987 .

12. SPRUNT, B., LEHOCZKY, J. and SHA, L., **Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm,** Proc. Real Time Systems, Symp. San Diego, CA, December 1988 .

13. BAKER, T.P., **Stack-Based Scheduling of Real-Time Processes,** Proc. of IEEE Real-Time Systems Symp., December 1990.

14. LEHOCZKY , J.P.and RAMOS-THUEL,S., **An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems,** Proc. of IEEE Real-Time Systems Symp., December 1992.

15. AUDSLEY, N.C., BURNS, A., RICHARDSON, M.F. and WELLINGS, A.J., **Hard Real-Time Scheduling: The Deadline Monotonic Approach,** Proc. of 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, GA ,May 1991.

16. DAVIS, R.I., TINDELL, K.W. and BURNS, A., **Scheduling Slack Time in Fixed Pre-emptive Systems**, Proc. of IEEE Real-Time Systems Symp., December 1993.

17. CHETTO, H. and CHETTO, M., **An Adaptive Scheduling Algorithm for Fault-Tolerant Real-Time Systems**, THE SOFTWARE ENGINEERING JOURNAL, 6, 3,May 1991.

18. SILLY, M., CHETTO, H. and ELYOUNSI, N., **An Optimal Algorithm for Guaranteeing Sporadic Tasks in Hard Real-Time Systems**, Proc. of IEEE Symp. on Parallel and Distributed Systems, Dallas, TXS, December 1990.