Some Results of the Earliest Deadline Scheduling Algorithm

HOUSSINE CHETTO AND MARYLINE CHETTO

Abstract—**Task scheduling is an important issue in the design of a real-time computer system because tasks have execution deadlines that must be met, otherwise the system fails with severe consequences upon the environment. In this paper, we study the problem of scheduling periodic time critical tasks on a monoprocessor system. A periodic time critical task consists of an infinite number of requests, each of which has a prescribed deadline. Tasks are assumed to meet their timing requirements when scheduled by the Earliest Deadline algorithm and preemptions are allowed. We report results from some investigations into the problem of making optimum use of the remaining processor idle time in scheduling periodic tasks either as soon as possible or as late as possible. The major results consist of the statement and proof of properties relating to localization and duration of idle time intervals and enable us to provide an efficient algorithm for determining maximum quantity of total idle time available between any two instants. We describe how these results can be applied, first to the decision problem that arises when a sporadic time critical task occurs and requires to be run at an unpredictable time and second, to the scheduling problem that arises in a fault tolerant system using the deadline mechanism for which each task implements primary and alternate algorithms.**

Index Terms—**Deadline mechanism, idle time, preemptive scheduling, real-time.**

I. Introduction

CURRENTLY, the field of real-time scheduling is the focus of a great deal of research interest. This is due to the very frequent use of digital computers in real time applications, growing sophistication in real-time software for the last few years, and an increased interest in improving system performance and reliability.

In real-time monitoring and control applications, the controlling system built around one or more computers is integrated into its environment and acts on line upon it through an appropriate instrumentation so as to lead it into a desired state. The controlling system which amounts to a closed loop one, accepts data from the sensors, generally at regular intervals, processes the data following a predefined law, and outputs results to the environment via the actuators. Debugging real-time software from the specifications of a given application amounts first to inventoring all the events (interrupt signals) and states (measures) that express relations between the controlling system and the environment and second to defining actions as algorithms in order to ensure the intended control of the environment. These algorithms implemented in memory of the computer are generally called tasks. Real-time software then consists of a set of tasks and is characterized by its interactions with external devices and the timing constraints relating these interactions. Usually, timing constraints are described in terms of deadlines by which computations of tasks must absolutely be met or the system will be considered to have failed. In many systems, and especially in embedded systems, danger to human life or simply damage to equipment makes such failures unacceptable. It follows that the main feature of a real-time system is to be supplied with a highly efficient task scheduler whose crucial part is to carefully manage the access to the processor unit so that all the tasks meet their timing requirements.

In this paper, we consider a scheduling problem in which a single processor computing system executes a set of tasks $\mathfrak{I}$, each of which consists of a periodic sequence of requests. $\mathfrak{I}$ will be called a periodic task set. More specifically, a task $T_i \in \mathfrak{I}$ demands periodically, every P_i units of time C_i units of computation time. We assume that the requests for each task arrive at the system at the beginning of request periods and that deadline for completion of the requested computation in each period coincides with the beginning of the next period. Hence, in our model, a periodic task set will be denoted as follows: $\mathfrak{I} = \{T_i(C_i, P_i), i = 1 \text{ to } n\}$. In this characterization, every task T_i makes its first request at time zero. Moreover, some time critical tasks said to be sporadic may arrive at the system and require to be run just once, at any moment. So, we define a sporadic task set as follows: $\mathfrak{T} = \{R_i(r_i, C_i, d_i), i = 1 \text{ to } m\}$. In this characterization, task R_i is released at time r_i, requires C_i units of time and a deadline occurs at d_i. So, in any time interval $[0, t]$, a periodic task set $\mathfrak{I}$ can be likened to one and only one sporadic task set $\mathfrak{T}$ where each task R_i corresponds to a particular request that occurs within $[0, t]$.

A periodic task set is said to be feasibly scheduled by a certain algorithm if the computation of each request can be completed prior to the arrival of the next request of the same task. Throughout our discussion, we assume that a preemptive scheduling discipline is employed. Thus, a request of C_i units of computation time can be satisfied by one or more quanta of time which sum to C_i. We will say that a periodic task set $\mathfrak{I}$ is schedulable if there exists at least one algorithm that can feasibly schedule $\mathfrak{I}$. For a schedulable periodic task set that is feasibly scheduled by

Manuscript received December 22, 1987; revised November 24, 1988. Recommended by E. Gelenbe.

The authors are with the Laboratoire d'Automatique de Nantes, U.A. au CNRS 823, ENSM 1, Rue de la Noe 44072 Nantes Cedex 03, France.

IEEE Log Number 8930131.

a certain algorithm, it is interesting to determine the particular time intervals called idle times during which the processor is not occupied. Idle times can then be recovered to process additional tasks.

In this paper, we investigate the problem of estimating localization and duration of idle times when tasks are scheduled according to the Earliest Deadline scheduling algorithm [14], [17]. Under Earliest Deadline, denoted ED, at each instant of time t, higher priority is assigned to the task whose deadline is closer to t. Two implementations of ED, respectively called EDS and EDL, are possible such that tasks are processed, respectively, as soon as possible and as late as possible. We will establish some specific properties of schedules produced by EDL and show how we can take advantage of them to achieve the best use of the computer.

The first illustrative example concerns a real-time system in which sporadic tasks may occur at unpredictable times and be required to be processed before a specified deadline. Such a system was the subject of studies reported in [15] and [19]. Our results enable us to conclude that a measure of the maximum quantity of processor idle time, available for processing a sporadic task can be obtained in assuming that periodic tasks are processed according to EDL from the arrival time of the sporadic task. Note that in practice all the tasks, periodic and sporadic ones, are always scheduled together according to EDS. We are interested in using EDL only to simulate a schedule and to derive a measure required for deciding whether the task can be accepted. For this method to work, we propose an acceptance condition that can be easily tested in line whenever a sporadic task occurs and enables us to guarantee a feasible execution for it, each time it is possible.

The second example is relative to a fault tolerant system which uses the Deadline mechanism [2], [7], [18]. Under this mechanism, two independent algorithms respectively called primary and alternate are provided for each task and in any request, either the primary or the alternate must run completely. Moreover, we mean to use the Last Chance strategy under which the alternate is scheduled after the primary. So, we show that applying EDL to alternates results in a maximum available time for processing primaries and enables us to dynamically reorganize the schedule in an efficient manner whenever a primary succeeds.

The paper is organized as follows. The next section introduces new terminologies and recalls some background material. In Sections III and IV, we state some properties about schedules produced by EDS and EDL. Section V shows how these results can be used to provide an efficient solution to some specific problems. The paper concludes with a summary in Section VI.

II. Terminology and Background

In a given schedule, if at some time t there is no ready task to be run, we refer to the time span between the completion of the last task to be processed before t and the first task to be processed after t, as an idle time. In order to compare idle time length in schedules produced by different scheduling algorithms for a given task set, we define the availability function $f_Y^X(t)$ to be

$$f_Y^X(t) = \begin{cases} 1 & \text{if the processor is idle at } t \\ 0 & \text{else.} \end{cases}$$

$f_Y^X(t)$ is defined with respect to a task set Y, scheduled according to the scheduling algorithm X in the time interval $[0, t]$. So, for any instants t_1 and t_2, the integral $\int_{t_1}^{t_2} f_Y^X(t)\,dt$ gives the total units of time that the processor is idle in the time interval $[t_1, t_2]$. We shall use $\Omega_Y^X(t_1, t_2)$ to denote this quantity.

A number of authors have studied the problem of devising algorithms for scheduling time critical tasks on a single processor computing system (see [1] and [6] for surveys). It was shown in [10], [14], and [17] that ED is optimal in the sense that all schedulable task sets can be feasibly scheduled by it. Performance evaluation of ED does not take into account the overheads originating from the task preemption. Furthermore, it supposes that there is no resource conflict between the concurrently executing tasks. This algorithm which is preemptive will feasibly schedule a periodic task set as long as its utilization factor U verifies:

$$U \leq 1 \text{ with } U = \sum_{i=1}^{n} \frac{C_i}{P_i}.$$

The fundamental property of the schedule produced by any preemptive algorithm and in consequence by ED, for a periodic task set is its cyclicity [12]. Let $P = \text{LCM}(P_1, P_2, \cdots, P_n)$ the base period be equal to the least common multiple of the periods $P_1, P_2, \cdots, P_n$. Let us assume $\mathfrak{I}$ is feasibly scheduled according to ED. This property means that the processor does exactly the same thing at time $t \geq 0$ that it does at time $t + kP$ $(k = 1, 2, \cdots)$. 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 $[kP, (k+1)P]$, $k = 0, 1, 2, \cdots$, each of them being called a window.

It comes that, for any instant t we have

$$f_{\mathfrak{I}}^{\text{ED}}(t) = f_{\mathfrak{I}}^{\text{ED}}(t + kP), \quad k = 1, 2, \cdots. \tag{1}$$

So, the schedule produced by ED for $\mathfrak{I}$ over an infinite time period can be considered as a time rigid one in the sense that all the requests of any task are started at a predetermined absolute global point in time. It comes that the remaining processor idle time, possibly required to process additional tasks is perfectly known thanks to function $f_{\mathfrak{I}}^{\text{ED}}(t)$ over the time interval $[0, P]$ and consequently over any window.

The most common way for implementing ED consists in ordering all ready tasks by increasing deadline (ties are broken arbitrarily) and executing them as soon as possible. ED is obviously an on line algorithm since it solely requires timing characteristics of tasks that are ready at any current time. Such implementation of ED will be called *e*arliest *d*eadline as *s*oon as possible and will be

denoted EDS. By opposition, we may imagine an implementation of ED that amounts to executing tasks as late as possible. For this case, we will show in Section IV that determination of the start time for any request requires preliminary construction of the schedule within $[0, P]$. This makes ED of an off line algorithm that will be called *e*arliest *d*eadline as *l*ate as possible and will be denoted EDL.

In the following sections, we will deduce some properties about functions f_3^{EDS} and f_3^{EDL} and show why these properties are fundamental, especially when attempting to solve specific problems as these previously evoked in Section I.

III. Main Results

In this section, we will prove three fundamental theorems which will be needed in our later discussion. Our goal is to deduce timing characteristics about schedules produced respectively by EDS and EDL for any sporadic task set and then, for any periodic task set.

In what follows $\mathfrak{Z}$ and $\mathbb{T}$, respectively, denote a periodic task set and a sporadic task set as defined in Section I. Let D denote the greatest deadline of sporadic tasks.

Theorem 1: Let X be any preemptive scheduling algorithm. For any instant t such that $t \leq D$,

$$\Omega_{\mathbb{T}}^{EDS}(0, t) \leq \Omega_{\mathbb{T}}^{X}(0, t). \tag{2}$$

Proof: We proceed to prove Theorem 1 by contradiction. We assume that there exists some algorithm X for which (2) does not hold. Let τ be the earliest time such that

$$\Omega_{\mathbb{T}}^{EDS}(0, \tau) = \Omega_{\mathbb{T}}^{X}(0, \tau) \tag{3}$$

and for any time

$$t > \tau, \Omega_{\mathbb{T}}^{EDS}(0, t) > \Omega_{\mathbb{T}}^{X}(0, t). \tag{4}$$

From (3) and (4) it follows that for any length Δ and in particular $\Delta \to 0$,

$$\Omega_{\mathbb{T}}^{EDS}(\tau, \tau + \Delta) > \Omega_{\mathbb{T}}^{X}(\tau, \tau + \Delta). \tag{5}$$

During the interval $[0, \tau]$, $\mathbb{T}$ has received an identical number of computation times by EDS and X because of (3). At time τ, two situations are possible:

- there is no ready task to be processed. Obviously, it follows that $\Omega_{\mathbb{T}}^{EDS}(\tau, \tau + \Delta) = \Omega_{\mathbb{T}}^{X}(\tau, \tau + \Delta) = \Delta$, in contradiction to (5).
- there is at least one ready task to be processed. As tasks are processed as soon as possible according to EDS, this means that the processor is working from τ up to $\tau + \Delta$ and so, $\Omega_{\mathbb{T}}^{EDS}(\tau, \tau + \Delta) \leq \Omega_{\mathbb{T}}^{X}(\tau, \tau + \Delta)$ in contradiction to (5). □

Theorem 2: Let X be any preemptive scheduling algorithm. For any instant t such that $t \leq D$,

$$\Omega_{\mathbb{T}}^{EDL}(0, t) \geq \Omega_{\mathbb{T}}^{X}(0, t). \tag{6}$$

Proof: We assume that there exists some preemptive algorithm X for which (6) does not hold. Let τ be the latest time such that

$$\Omega_{\mathbb{T}}^{EDL}(0, \tau) = \Omega_{\mathbb{T}}^{X}(0, \tau) \tag{7}$$

and for any time

$$t < \tau, \Omega_{\mathbb{T}}^{EDL}(0, t) < \Omega_{\mathbb{T}}^{X}(0, t). \tag{8}$$

From (7) and (8), it follows that, for any length Δ and in particular $\Delta \to 0$,

$$\Omega_{\mathbb{T}}^{EDL}(\tau - \Delta, \tau) < \Omega_{\mathbb{T}}^{X}(\tau - \Delta, \tau). \tag{9}$$

During the interval $[0, \tau]$, $\mathbb{T}$ has received an identical number of computation times by EDL and X because of (7). At time τ, two situations are possible:

- there is no ready task to be processed. Obviously it follows that $\Omega_{\mathbb{T}}^{EDL}(\tau - \Delta, \tau) = \Omega_{\mathbb{T}}^{X}(\tau - \Delta, \tau) = \Delta$, in contradiction to (9).
- there is at least one ready task to be processed. As tasks are processed as late as possible according to EDL, this means that the processor is working from τ down to $\tau - \Delta$ and so, $\Omega_{\mathbb{T}}^{EDL}(\tau - \Delta, \tau) = 0$, in contradiction to (9). □

For the purpose of stating the following theorem, we introduce a new sporadic task set defined from $\mathbb{T}$ as follows: $\mathbb{T}' = \{R_j'(r_j', C_j', d_j'), j = 1 \text{ to } m\}$ such that, for each task R_j' we have $r_j' = D - d_j$, $d_j' = D - r_j$ and $C_j' = C_j$.

Theorem 3: For any instant t such that $t \leq D$,

$$f_{\mathbb{T}}^{EDL}(D - t) = f_{\mathbb{T}'}^{EDS}(t). \tag{10}$$

Proof: The theorem is proved by contradiction. Let τ be the earliest time such that $f_{\mathbb{T}}^{EDL}(D - \tau) \neq f_{\mathbb{T}'}^{EDS}(\tau)$ and for any time t in $[0, \tau[$, (10) holds. Assume that

$$f_{\mathbb{T}}^{EDL}(D - \tau) = 1 \tag{11}$$

and

$$f_{\mathbb{T}}^{EDS}(\tau) = 0. \tag{12}$$

Equation (12) means that, at time τ, the processor is passive. As tasks are processed as soon as possible, it is clear that all the tasks in $\mathbb{T}'$ released before τ have been processed completely at time τ. So, within $[0, \tau[$ the processor has been occupied during $\Sigma_{j:r_j < \tau} C_j$ units of time or $\Sigma_{j:d_j > D - \tau} C_j'$ units of time.

Equation (11) means that, at time $D - \tau$, there exists at least one ready task R_j being executed and that necessarily verifies $r_j \leq D - \tau$ and $d_j > D - \tau$.

From (10) and linearity of the integral, it follows that. for any length Δ with $\Delta \leq \tau$

$$\int_0^{\tau - \Delta} f_{\mathbb{T}'}^{EDL}(D - t)\, dt = \int_0^{\tau - \Delta} f_{\mathbb{T}'}^{EDS}(t)\, dt$$

which implies

$$\Omega_{\mathbb{T}}^{EDL}(D - \tau + \Delta, D) = \Omega_{\mathbb{T}'}^{EDS}(0, \tau - \Delta). \tag{13}$$

In particular, if $\Delta \to 0$, from (12) and (13) it follows that

$$\Omega_{\mathbb{T}}^{EDL}(D - \tau - \Delta, D) = \tau - \sum_{j:d_j > D-\tau} C_j.$$

This means that, within $]D - \tau, D]$ the processor was occupied during $\Sigma_{j:d_j>D-\tau} C_j$ units of time. This quantity corresponds to the total computation time required by all the tasks that may be processed within $]D - \tau, D]$, i.e., tasks R_j that verify $d_j > D - \tau$. As tasks are processed as late as possible, this means that, at time $D - \tau$, no such task has already started execution and so (11) cannot hold. □

Example: Let $\mathbb{T} = \{R_1(4, 2.5, 9), R_2(2.5, 2, 11.5), R_3(0, 3, 7.5)\}$, so $D = 11.5$. It follows that $\mathbb{T}'$ is deduced from $\mathbb{T}$ as follows: $\mathbb{T}' = \{R_1'(2.5, 2.5, 7.5), R_2'(0, 2, 9), R_3'(4, 3, 11.5)\}$. The result of Theorem 3 is illustrated in Figs. 1 and 2.

Corollary 1: Let X be any preemptive algorithm and $\mathfrak{I}$ be a periodic task set. For any instant t such that $t \le P$, we have:

1) $\Omega_{\mathfrak{I}}^{EDS}(0, t) \le \Omega_{\mathfrak{I}}^{X}(0, t)$
2) $\Omega_{\mathfrak{I}}^{EDL}(0, t) \ge \Omega_{\mathfrak{I}}^{X}(0, t)$
3) $f_{\mathfrak{I}}^{EDL}(P - t) = f_{\mathfrak{I}}^{EDS}(t)$.

Proof: Every periodic task $T_i(C_i, P_i)$ in $\mathfrak{I}$ can be modeled by the set of its requests that arrive within $[0, P[$. Let $\mathfrak{R}_i$ be this set and defined as follows:

$$\mathfrak{R}_i = \Big\{ R_j(r_j, C_j, d_j)/r_j = kP_i,\ d_j = (k+1)P_i,\ C_j = C_i,\quad k = 0, 1, \cdots, n_i - 1 \text{ with } n_i = \frac{P}{P_i} \Big\}.$$

Clearly, it follows that the periodic task set $\mathfrak{I}$ can be likenned to a sporadic task set $\mathbb{T}$, defined as follows:

$$\mathbb{T} = \bigcup_{i=1}^{n} \mathfrak{R}_i.$$

Let $m = \Sigma_{i=1}^{n} n_i$ and be the number of sporadic tasks in $\mathbb{T}$. We note that

$$\max_{1 \le j \le m} \{d_j\} = P \quad \text{and} \quad \min_{1 \le j \le m} \{r_j\} = 0.$$

So, scheduling $\mathfrak{I}$ by EDS over an infinite time period amounts to scheduling $\mathbb{T}$ by EDS over each window

$$[kP, (k+1)P[, \quad k = 0, 1, 2, \cdots.$$

Let $\mathbb{T}'$ be the sporadic task set associated to $\mathbb{T}$ and defined as it was previously. It follows that

$$\mathbb{T}' = \{R_j'(r_j', C_j', d_j')/r_j' = (n_i - k - 1)P_i,\ d_j' = (n_i - k)P_i,\ C_j' = C_i,\ k = 0, 1, \cdots, n_i - 1,\ i = 1, 2, \cdots, n\}.$$

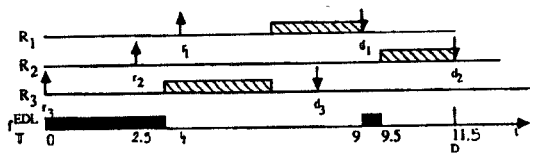

Fig. 1. Schedule produced by EDL on .

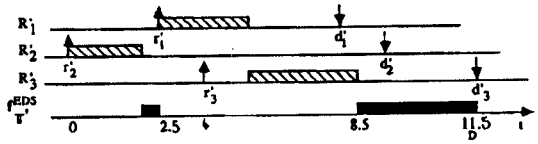

Fig. 2. Schedule produced by EDS on '.

Let $k' = n_i - k - 1$. Since $0 \le k \le n_i - 1$, we have $0 \le k' \le n_i - 1$ and $\mathbb{T}'$ can be defined as follows:

$$\mathbb{T}' = \{R_j'(r_j', C_j', d_j')/r_j' = k'P_i,\ d_j' = (k'+1)P_i,\ C_j' = C_i,\ k' = 0, 1, \cdots, n_i - 1,\ i = 1, 2, \cdots, n\}.$$

It is clear that

$$\mathbb{T} = \mathbb{T}'. \tag{14}$$

According to the previous theorems and from (14), proof of Corollary 1 becomes immediate. □

IV. Idle Time Determination

In this section, we investigate the problem of estimating localization and duration of idle times within a schedule respectively produced by EDS and EDL for $\mathfrak{I}$, over the time interval $[0, P]$ and consequently over any window $[kP, (k+1)P]$, $k = 1, 2, \cdots$.

A. Schedule Produced by EDS

Our goal here is to provide a simple way for determining localization and duration of idle times in any window of a schedule produced by EDS.

Let $E = \{r: r = kP_i, 0 \le k \le n_i, i = 1, 2, \cdots, n\}$. We construct a row vector $\mathcal{E}$ called arrival time vector, from the distinct elements of E. $\mathcal{E} = (e_0, e_1, \cdots, e_i, e_{i+1}, \cdots, e_p)$ with $e_i < e_{i+1}$, $e_0 = 0$ and $e_p = P$. We note that $p \le N - n + 1$ where N denotes the total number of distinct requests that occur within $[0, P[$. In [3], it was proved that, if an idle time exists, it precedes immediately an instant of $\mathcal{E}$.

So, let us evaluate for each time e_i, $1 \le i \le p$, the length of its idle time, denoted Δ_i and possibly equal to zero. By convention $\Delta_0 = 0$.

Within any time interval $[0, e_i[$, $i = 1$ to p, any task T_j is released $\lceil e_i/P_j \rceil$ times where $\lceil X \rceil$ denotes the smallest integer greater than or equal to X. So, if all tasks T_j have completed before e_i, it follows that

$$\Omega_{\mathfrak{I}}^{EDS}(0, e_i) = e_i - \sum_{j=1}^{n} \left\lceil \frac{e_i}{P_j} \right\rceil C_j$$

and consequently

$$\Delta_i = e_i - \sum_{j=1}^{n} \left\lceil \frac{e_i}{P_j} \right\rceil C_j - \sum_{k=1}^{i-1} \Delta_k; \text{ else } \Delta_i = 0.$$

In the general case, the length of the idle time that precedes any arrival time e_i is given by the following formulas:

$$\Delta_0 = 0$$

$$\Delta_i = \sup\left(0, e_i - \sum_{j=1}^{n} \left\lceil \frac{e_i}{P_j} \right\rceil C_j - \sum_{k=1}^{i-1} \Delta_k\right)$$

$$i = 1, 2, \cdots, p. \tag{15}$$

Now, the availability function $f_{\mathfrak{I}}^{EDS}$ is entirely defined by vector $\mathfrak{E}$ of distinct arrival times within $[0, P[$ and by the associated vector $\mathfrak{D}$ called idle time vector. $\mathfrak{D} = (\Delta_0, \Delta_1, \cdots, \Delta_p)$ where Δ_i is the length of the idle time that precedes e_i, with a length possibly equal to zero.

From definition of $\mathfrak{D}$, it follows that $\Sigma_{i=0}^{p} \Delta_i = \Phi$ where Φ denotes the quantity of processor idle time within any window and is given by the following formula:

$$\Phi = P(1 - U). \tag{16}$$

Since $f_{\mathfrak{I}}^{EDS}$ is cyclic with period P, $\mathfrak{D}$ and $\mathfrak{E}$ also define $f_{\mathfrak{I}}^{EDS}$ for any window. We note that description of the schedule over an infinite time is achieved in $O(N)$ operations which makes of this method an efficient one. We will show in Section V how it can be applied to the decision problem that arises when an additional time critical task arrives and is required to be run.

B. Schedule Produced by EDL

Let $\mathfrak{E}$ be defined as previously. Under EDS, any idle time within $[0, P]$ is situated just before a time that belongs to $\mathfrak{E}$. From Corollary 1, it follows that any idle time in a schedule produced by EDL is situated just after such a time.

Let us evaluate for each time e_i, $i = 0$ to p the length of idle time denoted Δ_i^* that directly follows e_i. By convention $\Delta_p^* = 0$.

Within any interval $[e_i, P]$ with $0 \le i < p$, every task T_j is released $\lceil P - e_i/P_j \rceil$ times. If all the ready tasks have completed their execution within $[e_i, P]$, the total idle time within $[e_i, P]$ is $(P - e_i) - \Sigma_{j=1}^{n} \lceil P - e_i/P_j \rceil C_j$. As the total idle time in $[e_{i+1}, P]$ is given by $\Sigma_{k=i+1}^{p} \Delta_i^*$, it follows that

$$\Delta_i^* = (P - e_i) - \sum_{j=1}^{n} \left\lceil \frac{P - e_i}{P_j} \right\rceil C_j - \sum_{k=i+1}^{p} \Delta_k^*;$$

$$\text{Else } \Delta_i^* = 0.$$

So, in the general case, the length of the idle time that follows any arrival time e_i is given by the following formulas:

$$\Delta_p^* = 0$$

$$\Delta_i^* = \sup\left(0, P - e_i - \sum_{j=1}^{n} \left\lceil \frac{P - e_i}{P_j} \right\rceil C_j - \sum_{k=i+1}^{p} \Delta_k^*\right)$$

$$i = 0, 1, \cdots, p - 1. \tag{17}$$

i	0	1	2	3	4	5	6
e_i	0	4.5	6	9	12	13.5	18
Δ_i	0	0	0.5	1	0.5	0	3
Δ_i^*	3	0	0.5	1	0.5	0	0

Fig. 3. Idle times under EDS and EDL.

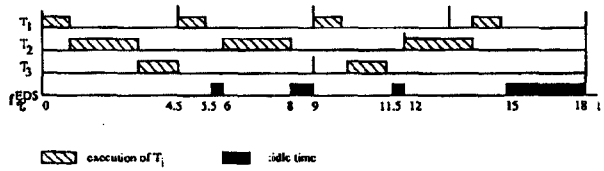

Fig. 4. Schedule produced by EDS for 3.

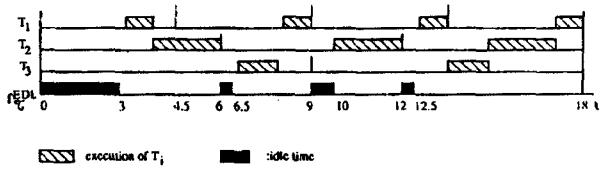

Fig. 5. Schedule produced by EDL for 3.

Now, let the idle time vector $(\Delta_0^*, \Delta_1^*, \cdots, \Delta_p^*)$ be denoted by $\mathfrak{D}^*$.

C. Illustration

Let $\mathfrak{I} = \{T_1(1, 4.5), T_2(2, 6), T_3(1.5, 9)\}$. $\mathfrak{I}$ is a schedulable periodic task set since $U = 0.722$. From construction of $\mathfrak{E}$ and thanks to formulas given previously, we deduce the length of any idle time within $[0, P]$ where $P = 18$ for the schedules respectively produced by EDS and EDL (see Fig. 3).

We note that $\Sigma_{i=0}^{6} \Delta_i = \Phi$ where $\Phi = 5$ and is given by (16). Besides, equality $f_{\mathfrak{I}}^{EDL}(P - t) = f_{\mathfrak{I}}^{EDS}(t)$ is expressed in the figure by symmetry, e.g., $\Delta_i^* = \Delta_{p-i}$. Results described in Fig. 3 are easily verifiable by constructing the schedules, produced by EDS (see Fig. 4) and EDL (see Fig. 5), respectively.

V. Applications

A. Acceptance of Sporadic Tasks

In real-time applications such as industrial process control, the computing system connected to the process is required to execute tasks in response to external signals and to guarantee that each such task will be completely processed before a specified deadline. In such a context and under normal functioning, software consists of a set of tasks that accept data and output set points to the control object via the actuators. Value of the task period then depends on the dynamics of the physical process it controls. Furthermore, in abnormal or critical situations such as a perturbation in the control object, sporadic tasks can be invoked. These additional tasks are also time critical tasks which occur at unpredictable times [15], [19].

In this section we shall be concerned with the algorithm which, given any occurring sporadic task R is capable of answering the question "Can R be accepted?." Notice

that R will be accepted if and only if there exists a valid schedule, i.e., a schedule in which R will execute by its deadline and periodic tasks will still meet their deadlines. We assume that, when any sporadic task arrives in the system and is required to be run, all the sporadic tasks previously accepted have completed their execution.

The question of the acceptance of a new task amounts to the question of the existence of a valid schedule. The answer then depends on timing parameters of all the tasks running on the intended machine such as computation times and deadlines. Besides these parameters, the scheduling strategy, i.e., the strategy of assigning the processor to the tasks and the amount of system overhead also affect the possibility of deadline missing and consequently the possibility of rejecting the new occurring task. In what follows, we propose a test able to answer "yes" to the question of the acceptance of a newly occurring task, each time it is possible. This means that any task rejected by this test would be rejected by any other test. The approach is based on the assumption that tasks are executed according to the optimal preemptive scheduling algorithm EDS.

Let τ be the current time which coincides with the arrival of a sporadic task R; upon arrival, task R is characterized by its release time r, its execution time C and its deadline d with $r + C \leq d$. Here, we assume that any occurring task is ready to be processed as soon as it arrives. Consequently, $r = \tau$. At the current time τ, the dynamic workload inflicted to the machine results from requests of periodic tasks. We can distinguish between current requests which have occurred at or before τ and have not completed at τ, and future requests which have not already started their execution at τ. Each of them corresponds to a pair of values of j and k such that either $kP_j \leq \tau < (k+1)P_j$ or $kP_j > \tau$ with $k > 0$. The pair (j, k) refers to the kth computation of the periodic task T_j. Each periodic task T_j is characterized by its static parameters C_j and P_j, by its dynamic execution time $C_j(\tau)$, e.g., execution time remaining at time τ and the deadline d_j of its current request at time τ.

Let $a = \lceil d/P \rceil$ and denote the number of successive windows that include $[\tau, d]$. Let $\mathfrak{I}(\tau)$ be the set of periodic requests that occur within $[\tau, aP]$. It is clear that feasible execution of requests in $\mathfrak{I}(\tau)$ and only these ones are possibly affected by the new occurring task. $\mathfrak{I}(\tau)$ can be considered a sporadic task set where each task (or request) has a release time equal or posterior to τ. Dynamic workload inflicted to the processor from τ is entirely described by $\mathfrak{I}(\tau)$ and enables us to ignore processor activity before time τ which becomes a new time zero. From Theorem 2, it is clear that applying EDL to $\mathfrak{I}(\tau)$ will result in the maximization of total idle time within any time interval $[\tau, t]$, $t \geq \tau$, and in particular within $[\tau, d]$. It follows that R will be accepted if and only if $C \leq \Omega^{\mathrm{EDL}}_{\mathfrak{I}(\tau)}(\tau, d)$.

Now, let us derive the value of $\Omega^{\mathrm{EDL}}_{\mathfrak{I}(\tau)}(\tau, d)$. For this purpose, we will need to use $I(\tau)$ and $A(\tau)$ respectively defined as to 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

$$\Omega^{\mathrm{EDL}}_{\mathfrak{I}(\tau)}(\tau, d) = a\Phi - A(\tau) - I(\tau) - \Omega^{\mathrm{EDL}}_{\mathfrak{I}(\tau)}(d, aP). \tag{18}$$

Computation of $\Omega^{\mathrm{EDL}}_{\mathfrak{I}(\tau)}(d, aP)$ is then achieved as follows.

Let $\mathcal{E}(\tau) = \{e_i(\tau)/e_i(\tau) = e_i + (a-1)P$ with $e_i \in \mathcal{E}\}$ and let h be the index that verifies $e_h(\tau) = \max\{e_i(\tau) \in \mathcal{E}(\tau)/e_i(\tau) < d\}$. Let $\Delta_i^*(\tau)$ denote duration of the idle time that follows the release time $e_i(\tau)$ by applying EDL to $\mathfrak{I}(\tau)$. By convention, $\Delta_p^*(\tau) = 0$. Let $M = \max_{\tau_j \in \mathfrak{I}(\tau)}\{d_j\}$. M denotes the maximum deadline among those of current requests of periodic tasks. In order to compute $\Omega^{\mathrm{EDL}}_{\mathfrak{I}(\tau)}(d, aP)$ efficiently, it is interesting to distinguish between the two following cases:

1) either $a > 1$ or ($a = 1$ and $d \geq M$).
2) $a = 1$ and $d < M$.

In the first case, computation times respectively required by $\mathfrak{I}$ and $\mathfrak{I}(\tau)$ within $[d, aP]$ are identical. It follows that $\Omega^{\mathrm{EDL}}_{\mathfrak{I}(\tau)}(d, aP) = \Omega^{\mathrm{EDL}}_{\mathfrak{I}}(d, aP)$. From periodicity of $f^{\mathrm{EDS}}_{\mathfrak{I}}$, it follows that

$$\Omega^{\mathrm{EDL}}_{\mathfrak{I}(\tau)}(d, aP) = \sum_{i=h+1}^{p} \Delta_i^* + \epsilon \tag{19}$$

with

$$\epsilon = \sup\left(0, e_h + \Delta_h^* - \left(d - (a-1)P\right)\right).$$

So this quantity can be easily computed at any current time if we assume $\mathcal{E}$ and $\mathfrak{D}^*$ to be available from the initialization time.

In the second case, the total computation time required by periodic tasks available within $[e_i(\tau), aP]$ is given by

$$\sum_{\substack{j=1 \\ d_j > e_i(\tau)}}^{n} C_j(\tau) - \sum_{j=1}^{n} \left\lfloor \frac{aP - \sup\left(e_i(\tau), d_j\right)}{P_j} \right\rfloor C_j.$$

Define index l such that $e_l(\tau) = \min\{e_i(\tau)/e_i(\tau) \geq M\}$. We note that for all $l \leq i \leq p$, the total computation time required by periodic tasks within $[e_i(\tau), aP]$ is given by

$$\sum_{j=1}^{n} \left\lfloor \frac{aP - e_i(\tau)}{P_j} \right\rfloor C_j \text{ also equal to } \sum_{j=1}^{n} \left\lfloor \frac{P - e_i}{P_j} \right\rfloor C_j.$$

It follows that:

$$\Delta_i^*(\tau) = \Delta_i^* \quad \text{for } i = l \text{ to } p, \tag{20}$$

$$\Delta_i^*(\tau) = \sup\left(0, (P - e_i) - \sum_{\substack{j=1 \\ d_j > e_i(\tau)}}^{n} C_j(\tau) - \sum_{j=1}^{n} \left\lfloor \frac{aP - \sup\left(e_i(\tau), d_j\right)}{P_j} \right\rfloor \cdot C_j - \sum_{k=i+1}^{p} \Delta_k^*(\tau)\right) \quad \text{for } i = h \text{ to } l-1 \tag{21}$$

and finally,

$$\Omega^{EDL}_{\mathfrak{I}(\tau)}(d, aP) = \sum_{i=h+1}^{p} \Delta_i^*(\tau) + \epsilon(\tau) \quad \text{with} \tag{22}$$

$$\epsilon(\tau) = \sup\left(0, e_h(\tau) + \Delta_h^*(\tau) - d\right).$$

The procedure that implements the acceptance test will need to use several data structures. The arrival time vector $\mathcal{E}$ and the static idle time vector $\mathfrak{D}^*$ are maintained in the Arrival Time Table (ATT) and the Static Idle time Table (SIT), respectively. Information on periodic tasks is maintained in a data structure called the Periodic Task Table (PTT). Each entry in the PTT contains a period, an execution time, a dynamic execution time, and a deadline. Tasks in PTT are ordered by their deadlines. Dynamic values $A(\tau)$ and $I(\tau)$, which are reinitialized at the beginning of every window, must be available for the procedure described in Fig. 6.

Clearly, this acceptance test runs in $O(N)$ time in the worst case. This makes the proposed approach an efficient one in the sense that few overheads are induced whenever a sporadic task arrives. Usually, this test was achieved according to one of the two following approaches:

- the approach proposed in [11]. The acceptance problem is then solved as a network flow problem in $O(N^3)$ time.
- the approach recently proposed in [4] which is an extension of the work presented in [8]. It consists in testing feasibility for requests that occur within $[\tau, aP]$ with deadline posterior to d. This test runs in $O(N^2)$ time.

Complexity of an on line algorithm is an evaluation of overheads that are produced when this algorithm runs, e.g., when a sporadic task occurs. So, in a hard real time context where all the tasks must imperatively meet their timing requirements, it is of more practical interest to make use of an algorithm which is both optimal in terms of scheduling performance and efficient in terms of computational complexity.

```
Algorithm ACCEPTANCE;
begin
  bool:=false;
  SOM:=0;
  for the last entry j of the PTT
    do
      M:=d_j;
      if a=1 and d<M then bool:=true end if
    end do
  find the last entry h in ATT such that e_h≤d-(a-1)P;
  if bool = false then
                  begin
                    for each entry j from the entry (h+1) to the last entry of the SIT
                    do SOM:=SOM+Δ*_j end do

                    compute ε from Δ*_h ;

                    SOM:=SOM+ε;
                  end
                else
                  begin
                    find the first entry l in ATT such that c_l≥M;
                    for each entry j from the last entry down to the entry l of the SIT
                      do SOM:=SOM+Δ*_j end do

                    for each entry j from the entry (l-1) down to the entry (h+1) of the SIT
                      do
                        compute Δ*_j(τ);

                        SOM:=SOM+Δ*_j(τ);

                      end do
                    compute Δ*_h and ε(τ);

                    SOM:= SOM+ε(τ);
                  end
  end if
  Ω^EDL_T(τ)(τ,d):=aΦ-A(τ)-I(τ)-SOM

  if C > Ω^EDL_T(τ)(τ,d) then return ('not accepted')
                  else return ('accepted')
  end if
end
```

Fig. 6. Outline of the acceptance test.

B. The Deadline Mechanism

Although carefully designed, every real-time system is subject to perturbations which result from subtle errors in software coding or failures due to hardware design deficiencies or malfunctions in input channels. In many systems and especially in embedded systems, such failures result in danger to human life or simply damage to equipment which is unacceptable. So, it has become necessary to design real-time systems that remain feasible, even in presence of failures, by implementing redundant software.

The Deadline mechanism, inspired by the recovery block scheme [16], allows us to construct a reliable computing system for hard real-time applications [7]. Each task implements on the one hand a program said primary that produces a good quality service but in an unknown length of time, due to possible failures and, on the other hand, a program said alternate that produces an acceptable result in a known and fixed length of time. In such a system, called a fault-tolerant real-time system, the occurrence of erroneous states does not result in timing failures and so, in any situation allows us to guarantee an acceptable although degraded performance.

In a controlling system that implements the Deadline mechanism, the scheduler must ensure that, within an infinite period, all the deadlines are met either by primaries or by alternates but in preference by primaries whenever it is possible. This goal can be achieved by applying the Last Chance strategy which runs as follows: the scheduler in charge of sequencing alternates reserves time intervals for processing them. For each request, this interval is chosen so that any alternate starts execution at the latest time. Primaries are then scheduled in remaining times before their alternate. Under this strategy, any alternate may interrupt a running primary at any instant for starting its execution at the correct time. But, whenever a primary successfully completes, execution of the alternate of the same request is no longer necessary. So, the system must be provided with a dynamic scheduler which reallocates the time reserved for this alternate so as to increase processor availability and consequently the number of primaries executed in the additional idle time.

So, an optimal scheduling algorithm produces a valid schedule whenever one exists and maximizes processor idle time so that primaries are processed as soon as possible.

Let $\mathfrak{I} = \{T_i(C_i, P_i), i = 1 \text{ to } n\}$ be the set of periodic tasks that run in such a system. For each task T_i, C_i, and P_i, respectively, denote the execution time of the alternate and the period of both alternate and primary. Up to date, few papers have been devoted to deduce theoretical performance measures for the deadline mechanism. The most significant result was obtained by Liestman and Campbell [13]. They have presented an optimal algorithm under the following assumptions:

- the period of each task is a multiple of the next smallest period.
- the execution times of primaries are known and fixed at system initialization.

Here, no such assumptions exist. Alternates are scheduled according to the Earliest Deadline algorithm and primaries by either preemptive algorithm. At the beginning of any window and in particular at system initialization time, the time intervals reserved for processing primaries in the current window are determined in assuming that alternates are scheduled according to EDL. From Theorem 2, this method enables us to state that primaries have a maximum available time for executing as soon as possible. However, when a primary completes successfully, a new schedule must be computed in order to recover the time interval previously affected to its alternate.

Let τ be the current time and coincide with the successful completion of a primary. The scheduling problem that arises in such system at time τ consists of reorganizing the schedule because execution of the associated alternate is no longer required. Such a reorganization is achieved after removing this alternate and by scheduling remaining alternates according to EDL from time τ up to the end point of the current window. Dynamic construction of the new schedule at τ is independent of processor activity before τ which can be considered a new time zero. Theorem 2 then allows us to conclude that the length of processor idle time (e.g., time available for processing primaries) in any interval $[\tau, t]$, $t \geq \tau$ is maximized and consequently the schedule so constructed is optimal.

Let $\mathfrak{D}^*(k)$ denote the dynamic idle time vector which results from the reorganization of the schedule following the success of the kth primary at time τ. Let d be the deadline of the current request of this primary and $\mathcal{E}(\tau)$ be defined as follows: $\mathcal{E}(\tau) = \{e_i(\tau)/e_i(\tau) = e_i + (a - 1)P\}$ where $a = \lceil d/P \rceil$.

By definition, $\mathfrak{D}^*(0) = \mathfrak{D}^*$ and $\mathfrak{D}^*(k) = \{\Delta_i^*(k)/e_i \in \mathcal{E}\}$ where $\Delta_i^*(k)$ is the length of the idle time that follows $e_i(\tau)$. By convention $\Delta_p^*(k) = 0$. Let $k = \min\{i/e_i(\tau) \geq \tau\}$ and $l = \max\{i/e_i(\tau) < d\}$. For each T_j, let d_j be defined as previously and b_j be equal to zero if primary of the current request has succeeded and equal to one otherwise. For each pair (T_j, e_i), let

$$S_{ij} = \begin{cases} 1 & \text{if } d_j > e_i(\tau) \\ 0 & \text{else.} \end{cases}$$

Computation of $\mathfrak{D}^*(k)$ from $\mathfrak{D}^*(k-1)$ is achieved efficiently after noting that alternates, which at step $(k-1)$ have been scheduled after d, remain identically scheduled after d at step k. Then it can be easily stated that:

$$\Delta_i^*(k) = \sup\left(0, (P - e_i) - \sum_{j=1}^{n}\left(\left\lceil \frac{P - \sup(e_i(\tau), d_j)}{P_j} \right\rceil + S_{ij}b_j\right) \cdot C_j - \sum_{j=i+1}^{p} \Delta_j^*(k)\right) \quad \text{for } i = l \text{ to } 1 \tag{23}$$

and

$$\Delta_i^*(k) = \Delta_i^*(k-1) \quad \text{for } i = l+1 \text{ to } p. \tag{24}$$

Let ATT, SIT, and PTT be the tables defined as previously. To the entry j of PTT, is associated the additional value b_j. Let DIT be the Dynamic Idle time Table that is updated whenever a primary succeeds. The global scheduler implements a cyclic algorithm with period P (see Fig. 7). At any time t it amounts to either an alternate scheduler or a primary scheduler. To determine invocation times of the procedure that updates DIT the system uses a boolean function SUCCES defined as follows:

$$\text{SUCCES}(\tau) = \begin{cases} \text{true} & \text{if time } \tau \text{ coincides with the successful completion of a primary} \\ \text{false} & \text{else.} \end{cases}$$

The main characteristic of this scheduling algorithm lies in the ability to dynamically specify scheduling intervals reserved for alternates, taking into account the run time situations, e.g., success or failure of primaries. As alternates are always scheduled as late as possible, the adaptivity of the proposed strategy makes it an optimal one. Its use is expected in applications that require high reliability and graceful degradation without needing human intervention. An extension to be suggested consists of proposing a practical implementation of this scheduling method in distributed systems. Up to date, strategies for allocating primaries and alternates to the nodes in such a system have been proposed [9]. However, it remains to be determined how a distributed scheduler can work.

```
Algorithm SCHEDULE
begin
  initialize DIT with SIT;
  for each entry j from the first entry to the last entry of the ATT
    do
      while τ<e_{j+1}(τ)
        do
          if τ<e_j(τ)+ Δ*_j(k) then
                        begin
                          schedule primaries by either algorithm;
                          if SUCCES (τ) then
                            begin
                              find the smallest entry k in ATT such that e_k(τ)≥τ;
                              find the greatest entry l in ATT such that e_l(τ)<d;
                              for each entry i from the entry l down to the entry k of the DIT
                              do
                                  compute new Δ*_i(k);
                              end do
                            end
                          endif
                        end
                      else schedule alternates by EDS;
          endif
        end do
    end do
end
```

Fig. 7. Outline of the dynamic scheduler.

VI. Summary

One of the most important steps in designing a real time computer system is to supply it with an efficient task scheduler. In a real-time context, efficiency is essential both for achieving the best use of the computer and for adhering with severe timing constraints relating to task executions.

One could think that preemptive scheduling is less suitable than nonpreemptive scheduling, whose implementation is simple and involves little overhead. However, the problem of determining a nonpreemptive optimal schedule is known to be NP-hard.

So, our aim was to bring to light new ideas about preemptive scheduling applied to a set of periodic tasks that run on a monoprocessor machine. More precisely, statements and proofs of fundamental properties about the schedule produced by the optimal Earliest Deadline algorithm were given.

Objectives of our theoretical investigations were to improve system performance and reliability in providing efficient solutions to various practical problems as described in Section V of this paper. Although we confined our attention to scheduling independent tasks on a monoprocessor machine, we believe that an extension of the model will make it more accurately reflect real-time programming. So, our plan for the future is to encompass a similar theoretical study for a multiple processor system by taking into consideration precedence and resource constraints.

References

[1] J. Blazewicz, "Deadline scheduling of tasks—A survey," *Foundation Contr. Eng.*, pp. 203-216, 1977.

[2] R. H. Campbell, K. H. Horton, and G. G. Belford, "Simulation of a fault-tolerant deadline mechanism," in *Proc. FTCS-9*, 1979, pp. 95-101.

[3] H. Chetto and M. Chetto, "On the acceptation of non-periodic time critical tasks in distributed systems," in *Proc. 7th IFAC Workshop Distributed Computer Control Systems (DCCS-86)*, Maychoss, West Germany, Oct. 30-Sept. 2, 1986.

[4] —, "How to insure feasibility in a distributed system for real time control," in *High Performance Computer Systems*, E. Gelenbe, Eds. Amsterdam, The Netherlands: North-Holland.

[5] M. R. Garey and D. S. Jonhson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[6] M. J. Gonzalez, Jr., "Deterministic processor scheduling," *ACM Comput. Surveys*, vol. 9, no. 3, pp. 173-203, Sept. 1977.

[7] H. Hecht, "Fault tolerant software for real-time applications," *ACM Comput. Surveys*, vol. 8, no. 4, pp. 391-407, 1976.

[8] W. Horn, "Some simple scheduling algorithms," *Naval Res. Logist. Quart*, vol. 21, pp. 177-185, 1974.

[9] C. M. Krishna and K. G. Shin, "On scheduling tasks with a quick recovery from failure," *IEEE Trans. Comput.*, vol. C-35, no. 5, pp. 448-455, May 1986.

[10] J. Labetoulle, "Some theorems on real time scheduling," in *Computer Architectures and Networks*, E. Gelenbe and R. Mahl, Eds. Amsterdam, The Netherlands: North-Holland, pp. 285-298.

[11] E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors," *Inform. Processing Lett.*, vol. 12, no. 1, pp. 9-12, Feb. 1981.

[12] J. Y. T. Leung and M. L. Merril, "A note on preemptive scheduling of periodic real-time tasks," *Inform. Processing Lett.*, vol. 20, no. 3, pp. 115-118, 1980.

[13] A. L. Liestman and R. H. Campbell, "A fault-tolerant scheduling problem," *IEEE Trans. Software Eng.*, vol. SE-12, no. 11, pp. 1089-1095, Nov. 1986.

[14] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, 1973.

[15] K. Ramamritham and J. Stankovic, "Dynamic task scheduling in hard real-time distributed systems," *IEEE Software*, vol. 1, no. 3, pp. 65-75, July 1984.

[16] B. Randell, P. A. Lee, and P. C. Treleaven, "Reliability issues in computing system design," *ACM Comput. Surveys*, vol. 10, no. 2, pp. 123-165, 1978.

[17] O. Serlin, "Scheduling of time critical processes," in *Proc. Spring Joint Computer Conf.*, 1972, pp. 925-932.

[18] A. Y. Wei, K. Hiraishi, R. Cheng, and R. H. Campbell, "Application of the fault-tolerant deadline mechanism to a satellite on-board computer system," in *Proc. FTCS-10*, 1980, pp. 107-109.

[19] W. Zhao and K. Ramamritham, "Distributed scheduling using bidding and focussed addressing," in *Proc. Real-Time Systems Symp.*, San Diego, CA, Dec. 3-6, 1985, pp. 112-122.

Houssine Chetto was born in Berkane, Morocco, on December 15, 1952. He received the Maitrise d'Electronique, Electrotechnique et Automatique from the University of Orléans, France, in 1978 and the degree of Docteur de 3ème cycle in control engineering from the University of Nantes, France, in 1981.

Between 1981 and 1985, he was an Assistant Professor in the Department of Physics of the University of Fès, Morocco. From October 1985 to September 1988, he has been employed by Ecole Nationale Supérieure de Mécanique de Nantes as an Assistant Professor in the Department of Automatic Control. He is presently an Assistant Professor in the Department of Production Management, Institut Universitaire de Technologie, Nantes. Since 1979, he has been working with the Laboratoire d'Automatique de Nantes. His fields of interest are concurrent computing, fault-tolerance, and scheduling, with a particular focus on real-time systems.

Maryline Chetto was born in Illiers, France, on December 18, 1959. She received the Maitrise de Sciences et Techniques in electrical engineering and the degree of Docteur de 3ème cycle in control engineering from the University of Nantes, France, in 1981 and 1984, respectively.

From October 1984 to December 1985, she held the position of Assistant Professor of Computer Science at the University of Rennes, while her research was with the Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes. In 1986, she joined the University of Nantes and is currently an Assistant Professor with the Institut de Recherche et d'Enseignement Supérieur aux Techniques de l'Electronique. From 1982 to 1984 and since 1986, she has been working at the Laboratoire d'Automatique de Nantes. Her research interests include scheduling, software fault-tolerance, and real-time operating systems.