How To Ensure Predictable Response Times for Real-time Tasks With Resource Constraints

Maryline Silly
Ecole Centrale de Nantes
Université de Nantes
LAN / U.A CNRS n° 823
1 rue de la Noé
44072 Nantes Cédex 03
FRANCE

Abstract: To meet its timing requirements, a real-time system must adopt a scheduling algorithm that permits to ensure a predictable response time. In this paper, we are specifically concerned with the problem of serving soft sporadic tasks in a uniprocessor system in which hard periodic tasks are scheduled using a dynamic priority algorithm. Typically, soft sporadic tasks benefit from being executed as early as possile while periodic tasks need to meet their deadlines. First, we present an analysis of preemptive Earliest Deadline scheduling for a model of independent periodic tasks.
Then, we augment our analysis to cater for periodic tasks which exclusively access to critical sections handled by the Kernelized Monitor protocol [9]. Our analysis determines the maximum processing time which may be stolen from periodic tasks without jeopardizing both their timing constraints and resource consistency. It provides the basis for an optimal dynamic scheduling algorithm in a context of resource sharing. Optimality means that every soft sporadic task is executed with a minimal response time.

Keywords: Real-time systems, Dynamic scheduling, Hard deadline periodic tasks, Sporadic tasks, Resource constraints

Maryline Silly was born at Illiers, France, in December 1959. She received the Maitrise d'Electronique from the University of Nantes, France, in 1981, the degree of Docteur de 3eme cycle in Concurrent Computing in 1984, and the degree of Habilité a Diriger des Recherches in 1993, respectively.
Between 1984 and 1985, she was an Assistant Professor at IFSIC (Institut de Formation aux Sciences de l'Informatique et de la Communication) of Rennes, France. Since 1986, she has been an Assistant Professor at IRESTE (Institut de Recherche et d'Enseignement aux Sciences et Techniques de l'Electronique) of Nantes and a researcher at the Laboratoire d'Automatique de Nantes , Ecole Centrale de Nantes. Her research interests are scheduling and fault-tolerance in real-time systems.

1. Introduction

Hard real-time systems are defined as being those systems in which a failure to meet the timing constraints imposed by the environment will result in property damage or even in a loss of human life. Whereas a large proportion of real-time systems implemented in the past were static, the new generation of applications required real-time software that might change during their life time. In the so -called dynamic real-time systems, no a priori knowledge of the task set is complete and consequently the effective sequence in which the tasks will take place cannot be determined off-line. Obviously, this implies that a scheduler has been designed to be able to cope with such dynamic change in processor workload. At any time during the lifetime of the application, there must be effected a re-evaluation of the timing behaviour of tasks for the future. This operation needs be relatively fast in order to reduce the overhead incurred by its execution.

In this paper, we address the problem of jointly scheduling periodic tasks with sporadic tasks. Timing constraints of periodic tasks are described by hard deadlines assumed to be less than or equal to their periods. A sporadic task consists of a computation that responds to internal or external events. It has a well-defined maximum execution time and a start time equal to its arrival time. Furthermore, periodic and sporadic tasks may indirectly interact with one another by sharing data or special devices (called resources) other than CPU. Consequently, we assume that they may have mutual exclusion requirements modelled as critical sections. Soft sporadic tasks benefit from being executed as soon as possible after their arrival.

We will present a dynamic uniprocessor scheduling algorithm that provides the shortest response time for every sporadic task. In contrast to [5] and [7] where tasks are scheduled using a

fixed priority algorithm, the now presented algorithm is a dynamic priority one and is based on the Earliest Deadline (ED) policy. The main results of this research concern the extension of specific properties of the Earliest Deadline algorithm stated in [3] to a more general periodic task model.

We will show how to determine the location and duration of processor idle times for any arbitrary window of time, which enables us to provide the maximum processing time which may be stolen from periodic tasks without jeopardizing their timing constraints. Then, we extend our approach to tasks that may exclusively access critical sections. We assume that the kernelized monitor protocol (KMP) is used. Recall that KMP is identical to ED with the restriction that all critical sections are non-preemptable. It can easily be proved that this scheduler prevents both deadlock and chained blocking since a task can only be blocked before its execution is started and since one active critical section can only exist at any time in the system. We show that a simple extension of the previous algorithm can accommodate resource constraints and determine the maximum processing time which may be stolen. Finally, we present a dynamic scheduling algorithm that services sporadic requests by making any spare processing time available as soon as possible.

The remainder of this paper is organized as follows. The next Section contains background material. Section 3 describes the task model under consideration. In Section 4, we report fundamental properties of the ED algorithm with task independence assumed and in Section 5, we show how to extend these properties in a context of resource sharing. Section 6 presents the dynamic algorithm for scheduling sporadic tasks and the paper concludes with Section 7.

2. Related Work

2.1 Scheduling Independent Tasks

The problem of jointly scheduling both hard and soft deadline tasks has been an active research area in the last few years. Most of approaches extend the Rate Monotonic algorithm and the Deadline Monotonic algorithm specifically designed to schedule periodic tasks with preemption allowed in static priority systems. The simplest approach consists in relegating soft tasks to background processing by executing them at a lower priority level than any hard periodic task. In another approach known as Polling, the capacity of a periodic task called server is used to service sporadic tasks. This pressupposes to compute off-line the capacity of this server such that the set of hard periodic tasks is schedulable. Other approaches termed Bandwith Preserving have been developed [6]. The Priority Exchange, Deferrable server and Sporadic server also give a preferential treatment of periodic tasks over sporadic tasks but they allow to preserve capacity throughout the server's period and not only at the beginning. While Bandwith Preserving methods lead to shorter response times than Polling and Background at low and medium loads, they degrade to provide the same performance as Polling at high loads. Furthermore, since they are based on the worst case execution time of periodic tasks, they do not permit to reclaim spare capacity when the effective execution time is less than the worst case execution time.

More recently, a new algorithm called Slack Stealing was developed by Lehoczky and Ramos Thuel [7]. It was proved to be optimal in the sense that it minimized the response time of soft sporadic tasks among all static priority algorithms which met deadlines of hard periodic tasks. The Slack Stealing algorithm consists in making any spare processing time available as soon as possible. Determination of processor idle time available at any instant is possible because the processor schedule is mapped out off-line and then inspected at run-time.

A variation of this algorithm, termed Dynamic Slack Stealing, was proposed by Davis et al [5] to permit to deal with a more general task model. Optimal for static priority systems, the Dynamic Slack Stealing algorithm computes the slack at run-time. Its high execution time overhead has led to developing approximate algorithms that provide close to optimal performance. Two similar approaches reported in [4] and [11] were designed for dynamic priority systems using the ED algorithm. They are based on an on-line computation of the maximum processing time

available for sporadic tasks, and execute them as soon as possible.

2.2 Scheduling Tasks With Resource Constraints

In many systems, tasks are not independent since they interact with each other by sharing resources such as data or devices. We usually call such resources that must be accessed exclusively by one task at a time, critical resources. Adding critical sections to real-time tasks makes the scheduling problem a NP-hard problem. If we use classical priority-driven policies, we must cope with a specific problem called priority inversion that occurs when a high priority task is forced to wait for the execution of many lower priority tasks for an indefinite length of time.

One way to reduce the priority inversion problem consists in using resource access control protocols which co-ordinate the access to shared resources: according to the so-called Kernelized Monitor protocol (KMP) [9], tasks are scheduled by ED and all critical sections are not preemptable. The Priority Ceiling protocol (PCP) proposed in [10] has been designed for systems with a fixed priority scheme. A priority ceiling is defined for every critical section and its value corresponds to the priority of the highest priority task which may enter the critical section. In the priority ceiling mechanism, a task T is allowed to enter a critical section only if its priority is higher than the priority ceilings of all critical sections currently used by any other task.

Similar to PCP, under the Dynamic Priority Ceiling Protocol (DPCP) described in [2], a priority ceiling is defined for every critical section associated with a shared resource. As it is based on the ED algorithm, the ceiling value at any time t is the priority of the highest priority task that may enter the critical section at or after time t. Baker [1] proposed another protocol called Stack Resource Policy (SRP) that could be used with either the RM or the ED algorithm. In addition to priority, every task is also assigned a fixed parameter called preemption level according to its period. The ceiling of critical section is defined to be the highest preemption level of those tasks that may enter the critical section. At any time, the current ceiling of the system is defined to be the maximum of the preemption level of the currently executed task and the ceilings of all locked critical sections.

Although the ceiling-based protocols have been extensively studied in the last few years, most of results only apply to periodic tasks and concern schedulability conditions.

3. System Characteristics

In this paper, we consider a set of n periodic tasks denoted by $\mathbf{T}=\{T_1,...T_n\}$ on a uniprocessor system. Each task T_i is characterized by its worst-case computation time C_i, a period P_i, and a deadline R_i measured relative to the time of the request and assumed to satisfy $R_i \leq P_i$. Tasks are indexed such that $i<j$ implies $R_i \leq R_j$. Then, each task gives rise to an infinite sequence of invocation requests, that begins at the time origin, here equal to zero. We assume that $\mathbf{T}$ is schedulable i.e. there exists at least one algorithm that can schedule the tasks of $\mathbf{T}$ such that their timing constraints should be met. The schedulability analysis can be realized, by constructing the schedule produced by ED and checking if the schedule is feasible. For $\mathbf{T}$, it suffices to map out the schedule over the hyperperiod of which length is equal to the least common multiple of the task's periods.

Further, each periodic task may share resources. The set of critical sections accessed by $\mathbf{T}$ is defined by $\mathbf{S} = \{S^i (B^i), i=1 \text{ to } m\}$ where B^i denotes the duration of critical section S^i. We assume that all requests of a task need enter the same set of critical sections. This is defined by a list CS_i for task T_i. A schedule for $\mathbf{T}$ is said to be *feasible* if all tasks in this set can be scheduled such that their timing constraints are met and the resources consistency is never violated. Here, KMP is used to handle resources.The worst case blocking time, which an invocation of task T_i can experience due to the operation of this protocol, will be denoted by B_i. To derive B_i, we need to identify the blocking set Z_i, which is the set of critical sections that can cause T_i to be blocked. Once Z_i is identified, B_i is the duration of the

longest critical section in Z_i. It can be stated that Z_i is defined as follows: $Z_i=\{s;\ s\in CS_k,\ R_k>R_i\}$.

The application software has also dynamic attributes due to the current execution of tasks which are available via the operating system and derived from data stored in a task control block. At any time, every hard periodic task is characterized by its remaining execution time and its deadline corresponding to the time instant by which its job must be completed.

4. Scheduling Independent Periodic Tasks

When a uniprocessor system solely supports periodic tasks, it is easy to know exactly what is done by the processor at any time since the schedule to be executed can be determined off-line and will not be modified during the life time of the application. Specifically it is interesting to determine the particular time intervals called idle times during which the processor is not occupied and that can be recovered to process additional tasks including soft and hard sporadic tasks.

With this end in view, we shall use $\Omega_Y^X(t_1,t_2)$ to denote the total units of time during which the processor is idle in the time interval $[t_1, t_2]$ when task set Υ is scheduled by algorithm X.

4.1 Basic Properties of ED

By virtue of executing soft sporadic tasks, we may imagine an implementation of ED that amounts to executing periodic tasks as late as possible without causing their deadline to be missed. Then, determination of the latest start time for every job of the periodic tasks requires preliminary construction of the schedule by the so-called Earliest Deadline as Late as possible algorithm denoted EDL.

Let $\Gamma=\{T_i\ (r_i, C_i, d_i),\ i=1 \text{ to } m\}$ be a set of independent hard deadline tasks (sporadic or periodic) and $D= \max\{d_i\}$. For every task T_i, it is respectively denoted by r_i, C_i, d_i, its release time, its time requirement and its deadline. We propose to recall a fundamental result which will be needed later in our discussion .

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

$$\Omega_\Gamma^X(0,t) \le \Omega_\Gamma^{EDL}(0,t) \qquad (1)$$

Proof: see [3]

Now, we may conclude that applying EDL to any hard task set will result in the maximization of total idle time within any time interval [0, t], $0 \le t \le D$.

4.2 Static Idle Times

Let us investigate the problem of estimating localization and duration of idle times within the schedule produced by EDL for Γ. We construct a row vector $\mathcal{K}$ called deadline vector, from a distinct request's deadlines within the hyperperiod plus the time instant 0. $\mathcal{K} = (k_0, k_1, ..., k_i, k_{i+1}, ..., k_q)$ with $k_i < k_{i+1}$, $k_0 = 0$ and $k_q = P - \inf\{x_i; 1 \le i \le n\}$ where $x_i = P_i - R_i$ for all $1 \le i \le n$ and $P=LCM(P_1,...P_n)$. Then, by using specific recurrent formulae reported in [4], we have a complete description of processor activity for the EDL schedule thanks to $\mathcal{K}$ and the associated idle time vector denoted $\mathcal{D}$ and equal to $(\Delta_0, \Delta_1,...,\Delta_i, \Delta_{i+1}, ..., \Delta_q)$ where Δ_i represents the length of the idle time which starts at time k_i .

Example 1.

Let consider the periodic task set Γ described by $(C_1, R_1, P_1)= (2, 6, 8)$, $(C_2, R_2, P_2)= (3, 11, 12)$ and $(C_3, R_3, P_3)= (4, 22, 24)$. The EDL

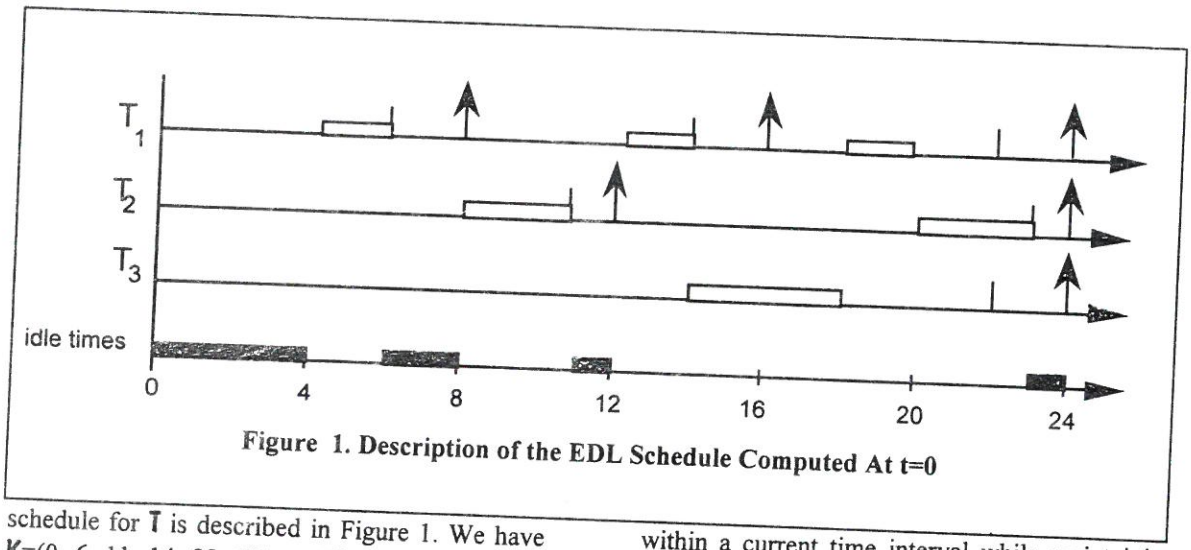

Figure 1. Description of the EDL Schedule Computed At t=0

schedule for $\mathbf{T}$ is described in Figure 1. We have $\mathbf{K}$=(0, 6, 11, 14, 22, 23) and $\mathbf{D}$=(4, 2, 1, 0, 0, 1) which means that the EDL busy periods are [4, 6], [8, 11] and [12, 23].

4.3 Dynamic Idle Times

Suppose we wish to determine the largest amount of processing denoted by $W_t(\Delta)$ that can be done during any time interval [t, t+Δ]. We assume that at time t, every periodic task is characterized by its dynamic attributes $C_i(t)$ and d_i where $C_i(t)$ is the maximum amount of processor time required for completing its current request, and d_i is the deadline of the request. We will denote by $\mathbf{T}(t)$ the set of periodic requests available from time t up to the end of the hyperperiod. $\mathbf{T}(t)$ is composed of current requests occurring at or before t and not being completed at t and of future requests which have not started yet their execution at t. By considering t as a new time reference, we are able to state corollary 2.

Corollary 2: *If* $\mathbf{T}$ *is a set of independent periodic tasks,* $W_t(\Delta) = \Omega^{EDL}_{T(t)}(t, t+\Delta)$ *for any instant t and any length Δ.*

Corollary 2 is fundamental because it stipulates that spare processor time, which can be recovered within a current time interval while maintaining adherence to timing constraints, is obtained by constructing the schedule where periodic tasks are executed according to EDL from the current time. On allowing for the dynamic occurrence of sporadic tasks at any time t, we know that the maximum amount of sporadic processing in [t, t+Δ] will be given by $\Omega^{EDL}_{T(t)}(t, t+\Delta)$.

The EDL schedule from time t is described by a *dynamic deadline vector* denoted $\mathbf{K}(t)$ and a *dynamic idle time vector* denoted $\mathbf{D}(t)$:

$$\mathbf{K}(t) = \left(k'_0, k'_1, \ldots k'_p\right)$$

with k'_0 = t and k'_1 = min{$k_i \in \mathbf{K}$; k_i>t}.

$$\mathbf{D}(t) = \left(\Delta'_0, \Delta'_1, \ldots \Delta'_p\right)$$

where Δ'_i denotes the length of the idle time that follows time k'_1 in the EDL schedule for $\mathbf{T}(t)$.

$\mathbf{D}(t)$ is computed by a recurrence relation which ends with Δ'_0 that represents the length of the time interval between t and the start time of the first busy period. This value is called system laxity at time t and is denoted by δ(t). Details of computation of $\mathbf{D}(t)$ are given in [4]. The first part of $\mathbf{D}(t)$ is computed from static values and

consequently, it requires 0(1) operations since we assume that the static idle time vector has been computed off-line and is available at any time. The second part of $\mathfrak{D}(t)$ requires at most O(K) operations where K represents the total number of requests which are released within a time interval of which length is limited by the greatest deadline R_i. In the worst case situation,

$$K = \sum_{i=1}^{n} \left\lceil \frac{\max\{R_j, 1 \le j \le n\}}{P_i} \right\rceil$$

where [x] denotes the least integer greater than or equal to x. Although the complexity of the approach is linear, it depends on the periods and deadlines of periodic tasks.

5. Scheduling Periodic Tasks With Resource Constraints

In this section, we relax the assumption that the periodic tasks are independent. Each request of a task T_i may access critical sections according to the Kernelized Monitor Protocol (KMP). Each request of T_i may therefore be blocked for at most B_i, the worst case blocking time.

5.1 Schedulability Analysis

In a context of resource sharing, the schedulability analysis is asked to ensure that all periodic tasks will always meet their deadlines and that consistency constraints of shared resources will never be violated. For these reasons, determining the schedulability of a task set requires an analytical approach which considers the properties of the scheduling algorithm and uses the timing information for the tasks. It can be proved that a sufficient condition for $\mathfrak{T}$ to be schedulable is the following:

$$\frac{C_1}{R_1} \ldots + \frac{C_i}{R_i} \ldots + \frac{C_n + B}{R_n} \le 1 \qquad (2)$$

where B is the execution time of the longest critical section. Condition (2) is derived by treating the worst case blocking delay B as an extra computation time of T_n in addition to its normal computation time C_n. However, blocking does not actually consume any processing time. Although this condition gives us a very simple test, it represents a pessimistic upper bound to processor utilization that guarantees schedulability.

In this section, we describe a more precise schedulability condition for KMP. The maximum amount of computation time needed for completing T_i and all higher priority tasks is given by

$\sum_{j=1}^{n} \left\lceil \frac{R_i + x_j}{P_j} \right\rceil C_j$ where [x] denotes the greatest integer less than or equal to x. Executing a critical section in the scheduling interval of T_i will affect all the tasks with a deadline greater than or equal to that of T_i.

Let $\mathfrak{L}_i$ be the ordered set of requests' deadlines within the time interval $[R_i, R_n]$.

Let $\delta^i = \min_{t \in L_i} \left(t - \sum_{j=1}^{n} \left\lceil \frac{t + x_j}{P_j} \right\rceil C_j \right)$. Clearly, δ^i represents a lower bound of additional computation time which the system can use within the scheduling interval of T_i while guaranteeing deadlines of lower priority tasks. We are now prepared to derive the following schedulability condition for KMP.

Theorem 3: *Using KMP, all tasks meet their deadlines if*

$$\sum_{i=1}^{n} \frac{C_i}{R_i} \le 1 \quad \text{and } B_i \ge \delta^i \quad \forall i,\ 1 \le i \le n \qquad (3)$$

Proof: For any task T_i, if neither T_i nor any of the higher priority tasks in I_i is blocked by lower priority tasks, condition $\sum_{i=1}^{n} \frac{C_i}{R_i} \le 1$ alone ensure that the task's deadline will always be met. If blocking does occur, let us prove that the

schedule is feasible if (3) is true. We prove this by contradiction. We assume that (3) is satisfied and the schedule is not feasible which means that there exists at least one task, T_k, which does not meet its deadline. This signifies that a task T_i, possibly T_k, such that $d_i \leq d_k$ was delayed by one lower priority task released before time r_k. Then, the maximum blocking delay of T_i is given by B_i.

The maximum processor time required by all higher priority tasks and T_k within the scheduling interval of T_k is given by $\sum_{j=1}^{n}\left\lceil \frac{R_k + x_j}{P_j} \right\rceil C_j$. As at most one task with a lower priority than T_i can provoke blocking for T_k, the worst case response time of T_k is $\sum_{j=1}^{n}\left\lceil \frac{R_k + x_j}{P_j} \right\rceil C_j + B_i$, necessarily greater than R_k since d_k is missed, which implies that

$$B_i > R_k - \sum_{j=1}^{n}\left\lceil \frac{R_k + x_j}{P_j} \right\rceil C_j \qquad (4)$$

Therefore $R_k \in L_i$ since $R_k \geq R_i$. As (3) is satisfied, in particular we have

$$B_i \leq R_k - \sum_{j=1}^{n}\left\lceil \frac{R_k + x_j}{P_j} \right\rceil C_j \qquad (5)$$

From (4) and (5), we have a contradiction.

5.2 Processor Idle Time With KMP

Although EDL is a useful algorithm, its application to tasks that access critical sections, is not intuitively obvious. With task independence assumed, EDL provides us a means for determining the maximum of spare processing time that can be available as soon as possible, without jeopardizing the hard timing constraints. We can prove that this algorithm extends without any modification for executing tasks as late as possible according to KMP from time zero because resource constraints do not interfere with processor activity, provided that the schedulability condition (3) is satisfied. Such a result can be proved by induction on the busy periods starting at the end of the hyperperiod. Consequently, we may disregard resource constraints for computing the largest amount of processing $W_0(\Delta)$ that can be done during $[0, \Delta]$ since it is given by $\Omega_T^{EDL}(0, \Delta)$.

Next we seek to determine the largest amount of processing, $W_t(\Delta)$, that can be done during any time interval $[t, t+\Delta]$. Let us consider that there is one resource accessed at time t. Indeed, we know that at most one such resource may exist. It is characterized by the deadline of the request that accessed it, d, and by the maximum amount of processor time required to unlock it, B(t). We will show that the schedule that maximizes processor idle time from time t up to the end of the hyperperiod, is obtained as follows:

- construct a new task set denoted $\Gamma'(t)$ by subtracting the execution time of the request with deadline d by B(t) and adding up the execution time of the most imminent request by B(t),

- construct the EDL schedule from t on $\Gamma'(t)$.

The resulting schedule will be called EDL_m schedule.

Now, we will show that the schedule produced by KMP in the EDL_m busy periods from time t is feasible and maximizes processor idle time as soon as possible. In this view, lemma 5 is of most interest.

Lemma 5: *If condition (3) is satisfied then the schedule produced by ED for $\Gamma'(t)$ in the EDL_m busy periods is feasible.*

Proof: We prove this by contradiction and suppose there is a task T_k with release time r_k and deadline d_k less than d which does not meet d_k in the schedule produced by ED for $\Gamma'(t)$. Consider the two following cases:

Case 1: there is at least one idle time period between t and d_k in the EDL_m schedule

Let denote by f the end time of this idle time period. This means that the total length of busy period from f to d_k is greater than or equal to the

sum of execution times for all the tasks of $\Gamma'(t)$ with a deadline less than or equal to d_k and greater than or equal to f. As T_k is ready to be processed from time f, T_k cannot miss its deadline, in contradiction with the hypothesis.

Case 2: there is no idle time period between t and d_k in the EDL_m schedule .

This means that the processor is continuously busy from r_k up to d_k executing T_k and tasks with a priority greater than T_k. The maximum processor time required by these tasks is given by $\sum_{j=1}^{n}\left\lfloor \frac{R_k + x_j}{P_j} \right\rfloor C_j + B_k$ where B_k is the longest critical section that can be executed in the scheduling interval of T_k . From Theorem 4, such quantity is assumed to be less than or equal to R_k which implies that d_k cannot be missed, in contradiction with the hypothesis.

Theorem 6: *The schedule produced by KMP in the EDL_m busy periods from time t is feasible.*

Proof: At time t, the dynamic workload imposed on the machine by $\Gamma(t)$ is composed of periodic requests which are available from t up to the end of the hyperperiod. Since requests with a deadline greater than d cannot be blocked by semaphores currently in $S(t)$, we know that the schedule produced by KMP in the EDL busy periods from time d up to the end of the hyperperiod is feasible since such a schedule is identical to the ED schedule.

Now, let prove that the schedule produced by KMP in the EDL_m busy periods from t up to d is feasible. From lemma 5, it suffices to prove that feasibility of ED for $\Gamma'(t)$ implies feasibility of KMP for $\Gamma(t)$.We prove this by contradiction and suppose ED is feasible for $\Gamma'(t)$ while there is a request of T_k with deadline d_k less than d which does not meet d_k assuming tasks be scheduled according to KMP.

The total length of the busy periods included in the scheduling interval I_k of T_k is sufficient to feasibly schedule by ED all the tasks of $\Gamma'(t)$ with a deadline less than or equal to d_k. Such tasks correspond to all the tasks of $\Gamma(t)$ which have a deadline less than or equal to d_k and an additional task which is characterized by its deadline equal to the deadline of the most urgent periodic request and its execution time equal to the amount of processor time required for leaving the critical section currently accessed at t. Executing tasks of $\Gamma'(t)$ according to ED within I_k then amounts to executing tasks of $\Gamma(t)$ according to KMP. As ED is feasible for $\Gamma'(t)$, KMP is also feasible for $\Gamma(t)$, which contradicts that d_k is missed.

Theorem 7:*The schedule produced by KMP in the EDL_m busy periods from time t maximizes processor idle time as soon as possible.*

Proof: We prove this by contradiction. We assume that there exists some scheduling algorithm X for which the schedule produced by KMP in the X schedule is feasible and there exists some instant t' such that

$$\Omega_{T(t)}^{EDL_m}(t,t') < \Omega_{T(t)}^{X}(t,t') \quad (6)$$

Let τ ($t \geq \tau$) be the latest time such that

$$\Omega_{T(t)}^{EDL_m}(t,\tau) = \Omega_{T(t)}^{X}(t,\tau) \quad (7)$$

From (6) and (7), it follows that

$$\Omega_{T(t)}^{EDL_m}(\tau,\tau+1) < \Omega_{T(t)}^{X}(\tau,\tau+1) \quad (8)$$

which means that the processor is busy from τ to $\tau+1$ in the EDL_m schedule while it is idle in the X schedule. During [t, τ], $\Gamma(t)$ has received an identical number of computation times by KMP in the EDL_m schedule and the X schedule respectively, because of (7). At time τ, two situations are possible:

Case 1: There is no ready task to be processed.

Obviously it follows that

$$\Omega_{T(t)}^{EDL_m}(\tau,\tau+1) = \Omega_{T(t)}^{X}(\tau,\tau+1) = 0,$$

in contradiction to (8).

Case 2: There is at least one ready task to be processed.

By the construction of the EDL_m schedule, we know that there exists a deadline d ($d>\tau$) such that d is followed by an idle time period and the processor is continuously busy from τ up to d executing tasks of $\mathbf{T}(t)$ with a deadline less than or equal to d and at most one critical section with a deadline less than d. As such tasks are processed as late as possible, inequality (8) implies that there is at least one task which cannot meet its deadline within $[\tau, d]$ in the X schedule, in contradiction with the hypothesis.

From Theorem 7, we are now able to compute $W_t^{KMP}(t')$ for any instants t and t' since it is given by $\Omega_{T(t)}^{EDL_m}(t,t')$ may there be some semaphore currently locked at t or not.

Example 2.

Let consider task set $\mathbf{T}$ of example 1 and suppose now that $CS_1=\{S^1\}$, $CS_2=\{S^2\}$ and $CS_3=\{S^1, S^2\}$ with $B^1=1$ and $B^2=2$. Condition (3) is satisfied and guarantees a feasible execution for $\mathbf{T}$ by KMP. Assume we wish to determine the maximum amount of processor idle time available between times 8 and 17. At time 8, critical section S^2 is being accessed. So, we compute the EDL schedule as described above (see Figure 2). We can verify that $\mathbf{K}(t)=(8, 11, 14, 22, 23)$ and $\mathbf{D}(t)=(3, 0, 4, 0, 1)$ and consequently $W_8(9)=6$. If tasks were independent, $W_8(9)$ would be equal to 7.

6. Considerations of Soft Sporadic Tasks

Next we introduce the soft sporadic tasks. Each sporadic task is specified by a processing requirement which corresponds to the maximum time required for its complete execution. Any sporadic task is ready to execute as soon as it arrives and there is no a priori knowledge about which set of sporadic tasks requests will be encountered. 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.

6.1 Sporadic Tasks With No Resource Constraints

First, assume that sporadic tasks do not share resources together with periodic tasks. Our scheduler will use a cyclic algorithm with period equal to the least common multiple of task periods. At the beginning of each hyperperiod the dynamic idle time vector is updated with the static idle time vector. When no sporadic task is pending for execution, periodic tasks are executed as soon as possible according to KMP. Whenever a sporadic task arrives while no other sporadic task was present, the dynamic idle time vector is computed, after checking whether a resource is currently accessed. Then, sporadic tasks are executed in idle times of the EDL_m schedule until they are all completed.

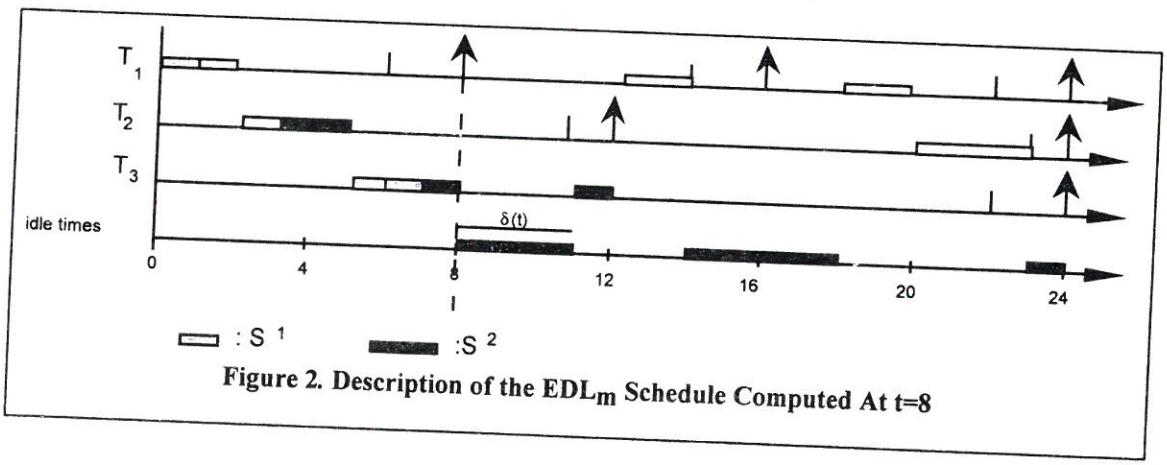

Figure 2. Description of the EDL_m Schedule Computed At t=8

6.2 Sporadic Tasks With Resource Constraints

We now consider the situation where sporadic tasks share one or more resources with periodic tasks. Suppose a sporadic task is being executed and requests to enter a critical section at current time t . To make sure that no deadline will be missed, the sporadic task must be guaranteed sufficient idle time to complete its critical section prior to the beginning of the next periodic task that will access this critical section. As we have no a priori knowledge about which task will access it, the task is only allowed to succeed in locking the resource if $\delta(t) \geq B$ where B is the length of the critical section and $\delta(t)$ the system laxity at t. Provided this condition holds and the requested resource is not currently accessed, sporadic tasks will be executed in idle times of the EDL schedule as long as they are all completed, or a sporadic task needs to lock a resource. If the critical section is currently accessed at t, the periodic task is executed first, just to unlock the resource, and sporadic tasks are then executed in the EDL idle time periods.

Assume that condition $\delta(t) \geq B$ is not verified. Let k be the next deadline followed by an idle time in the EDL schedule. Since the system laxity cannot increase as long as tasks with deadline k have not completed their execution, periodic tasks must be executed as soon as possible by KMP until their completion , say at time t_k. Then, the system laxity is given by $k-t_k + \Delta_i^t$ where Δ_i^t is the length of the idle time starting at time k and previously computed at t. And the locking condition is newly tested, leading to schedule periodic tasks as soon as possible or as late as possible depending on the test's result.

6.3 Comments and Illustration

We note that the time complexity of this strategy is comparable with that of any priority driven algorithm since the computation of the EDL schedule is only required when a sporadic task arrives while no other sporadic task was present. This means that computational overhead decreases as the load factor increases.

The optimality of the strategy lies in that periodic tasks are executed as soon as possible by the kernelized monitor protocol which enables us to optimize the processing power of the machine for the future. Whenever at least one sporadic task is pending for execution, periodic tasks are executed as late as possible in order to make this processing power available for sporadic tasks as long as they require to be run and as long as the system laxity is sufficient to execute critical sections non-preemptively.

Example 3.

Consider the previous periodic task set and assume that the first sporadic task, say T1, arrives at time 8. The dynamic idle time vector is immediately computed as described in Example 2. T_1 will be executed until it requires to enter critical section S^2 at time 10. As S^2 is currently accessed at time 10 and $\delta(10)=1$(so, less than B^2), periodic tasks must be executed first, until

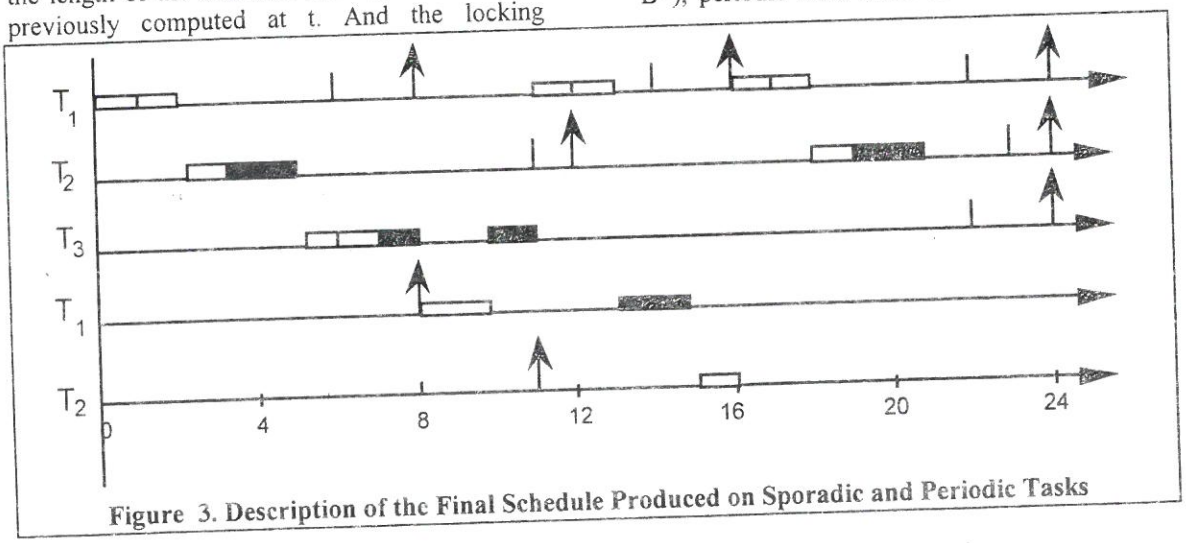

Figure 3. Description of the Final Schedule Produced on Sporadic and Periodic Tasks

all periodic requests with deadline 14 be completed since 14 is the earliest deadline followed by an idle time in the EDL schedule. At time 13, the system laxity gets higher enough so as to execute the critical section S^2 non-preemptively. T_1 finally completes at time 15. Task T_2 which arrived at time 11 is executed between times 15 and 16. The effective schedule produced within the hyperperiod is described in Figure 3.

We note that the response times of T_1 and T_2 are respectively 7 and 5, which provides a mean response time equal to 6. We can verify that, with a background strategy, T_1 and T_2 respectively complete at 20 and 21, which provides a mean response time equal to 11.

7. Conclusion

In this paper, we considered the problem of scheduling a set of periodic tasks initially assigned to a single processor machine, and in addition soft sporadic tasks that occur and require to be run on this machine at unpredictable times. We presented an analysis which builds upon the Earliest Deadline algorithm in order to schedule soft sporadic tasks as soon as possible while guaranteeing timing constraints of periodic tasks. It is an extension of our own work reported in [11] to tasks with mutual exclusion constraints, scheduled according to the Kernelized Monitor protocol.

By virtue of computing at run-time the maximum processor idle time that can be stolen, this algorithm has the great advantage of providing all the flexibility and predictability which are being increasingly required by next generation systems. These systems are highly evolutive and imply that the scheduler has been designed in such a way as to be able to cope with dynamic changes in processor workload. Furthermore, our approach can easily adapt to a more general class of scheduling problems including reclaiming unused periodic execution times and scheduling hard sporadic tasks. A similar study is currently developed to solve this problem when using the Dynamic Priority Ceiling Protocol among other ceiling-based protocols.

REFERENCES

1. BAKER, T.P., **Stack-based Scheduling of Real-time Processes**, IEEE Real-time Systems Symp., Dec.1990, pp. 191-200.
2. CHEN, M.I. and LIN,K.J., **Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-time Systems**, THE JOURNAL OF REAL-TIME SYSTEMS, Vol. 27, No. 4, Dec. 1990, pp.325-346.
3. CHETTO, H. and CHETTO-SILLY, M., **Some Results of the Earliest Deadline Scheduling Algorithm**, IEEE T.S.E, Vol. 15, No. 10, 1989, pp. 1261-1269.
4. CHETTO, H., **An Optimal Algorithm for Jointly Scheduling Sporadic and Periodic Tasks**, STUDIES IN INFORMATICS AND CONTROL, Vol. 4, No. 2, June 1995, pp 167-182.
5. DAVIS, R.I., TINDELL, K.W. and BURNS, A.,**Scheduling Slack Time in Fixed Priority Preemptive Systems**, IEEE Real-time Systems Symp., Dec. 1993, pp. 222-231.
6. LEHOCZKY, J.P., SHA, L. and STROSNIDER, J.K., **Enhanced Aperiodic Responsiveness in Hard Real-time Environments**, IEEE Real-time Systems Symp., Dec. 1987, pp. 166-171.
7. LEHOCZKY, J.P. and RAMOS -THUEL,S., **An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Fixed Priority Preemptive Systems**, IEEE Real-time Systems Symp., Dec.1992, pp. 110-123.
8. LIU, C. L. and LAYLAND, J.W., **Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment**, JACM, Vol. 20, No.1, 1973, pp. 46-61.
9. MOK, A.K., **Programming Language Support for Distributed Real-time Applications**, Technical Report, Department of Computer Sciences, University of Texas, Austin, 1987.

10. SHA, L., RAJKUMAR, R. and LEHOCZKY, J.P., **Priority Inheritance Protocols: An Approach To Real-time Synchronization**, IEEE T.C, Vol. 39, 1990, pp. 1175-1185.

11. SILLY, M.,**Un algorithme d'ordonnancement de tâches sporadiques pour les systèmes temps-réel**, Revue APII, Vol .28, No. 2 , 1994, pp. 179-205.