A Few Results on Non-Preemptive Real Time Scheduling

Laurent Georges, Paul Mühlethaler, Nicolas Rivierre

INRIA

INSTITUT NATIONAL DE RECHERCHJE EN INFORMATIQUE ET EN AUTOMATIQUE

A Few Results on Non-Preemptive Real time Scheduling

Laurent Georges, Paul Mühlethaler, Nicolas Rivierre

N° 3926

Mai 2000

THEME 1

Réseaux et système

ISSN 0249-6399 ISRN INRIA/RR--3926--FR+ENG

A Few Results on non-preemptive real time scheduling

Laurent Georges*, Paul Mühlethaler, Nicolas Rivierre**

THEME 1: Réseaux et système

Projet : HiPERCOM

Rapport de recherche n˚3926- MAI 2000

Abstract: In this paper, we investigate the non-preemptive scheduling problem as it arises in single processor systems. We focus on non-idling scheduling, the idling scheduling is briefly introduced in the lasts section. We extend some previously published results concerning preemptive and non-preemptive scheduling over a single processor. The main issue that we study in this article is the applicability and/or adaptation of results obtained in preemptive scheduling. In a first part we embark on revisiting aperiodic non idling non preemptive scheduling. We review complexity results and investigate conditions under which Earliest Deadline First is optimal in non preemptive scheduling. In a second part, we scrutinize periodic non idling non preemptive scheduling and we show that for non preemptive scheduling feasibility must be checked on a time interval of duration r+2P (r denotes the maximum of the release times and P the smallest common multiple of the task periods). We also show that a well established result concerning feasibility of task sets under non preemptive scheduling (no overload on any given time intervals) has no equivalence in non preemptive scheduling even if one takes into account the blocking factor. The third part is a very quick introduction to scheduling problems in an idling and non preemptive context.

Key-words: Non-preemptive scheduling, aperiodic tasks, periodic tasks, feasibility, complexity, optimality, EDF (Earliest Deadline First).

* Laurent Georges. IUT de Vitry / Dept GTR20, 120 rue Paul Armangot. 9400 Vitry sur Seine

** Nicolas Rivierre, CNET, 8 rue du Général Leclerc, 92794 Issy Moulineaux Cedex 9

Unité de recherche INRIA Rocquencourt
INRIA, Domaine de Voluceau, Rocquencourt BP 105 , 78153 Le Chesnay Cedex (France)
Téléphone : (33) 01 39 63 11 55- Télécopie : (33) 01 39 63 11 55

Quelques résultats en ordonnancement non préemptif

Cet article traite de l'ordonnancement non préemptif sur un mono-processeur. L'article concerne principalement l'ordonnancement non oisif bien que la dernière partie de l'article fournisse une introduction aux problèmes de l'ordonnancement non préemptif oisif. Il étend des résultats antérieurs concernant l'ordonnancement préemptif et non préemptif sur un mono-processeur. Le principal problème étudié dans cet article est l'applicabilité et/ou l'adaptation de résultats d'ordonnancement non préemptif. Dans une première partie, l'ordonnancement non oisif de tâches apériodiques est étudié ainsi que les conditions d'optimalité de la politique d'ordonnancement EPP (Echéance la plus Proche en Premier). Dans une seconde partie nous étudions l'ordonnancement non oisif et non préemptif de taches périodiques. Nous montrons d'abord que le test de faisabilité doit s'étendre sur une durée de r+2P où r est le dernier instant de génération d'un tache et P le plus petit commun multiple des périodes des taches en présence. Nous montrons aussi que le résultat bien établi de faisabilité en ordonnancement préemptif (pas de surcharge sur aucun intervalle) n'a pas d'équivalent en ordonnancement non préemptif même si l'on prend en compte le facteur de bloquage. La troisième et dernière partie de l'article présente une rapide introduction aux problèmes de l'ordonnancement oisif non préemptif.

Mots-clé : Ordonnancement non préemptif, tache périodique, tache aperiodic, faisabilité complexité, optimalité

1 Introduction

This paper addresses the problem of non-preemptive scheduling over a single processor. This problem has received less attention than preemptive scheduling, which has been extensively studied for the past thirty years. [LILA73], [LM80], [LEU82], [BHR90], [CC91] are an extremely small subset of existing publications on this subject. Works on non-preemptive scheduling are more recent and less abundant [LM80], [MC92], [JE91],[HOW95].

Although it can be shown that in non preemptive scheduling idling scheduling has a special interest, most of this article deals with non-idling scheduling. It is recalled that in non-idling scheduling, the processor can not be idle if there are released tasks pending.

The main difficulty in non preemptive scheduling is a direct consequence of the NP completeness of the scheduling problem. This NP completeness does not exist in preemptive scheduling since EDF (Earliest Deadline First) is optimal. The NP completeness is a known result in non-preemptive scheduling [GA79]. In this paper we will show a new result; this NP completeness whose existence has been proved for task set exhibiting a high load (generally 1.) can be obtained with task sets exhibiting a load which can be made less than any given number. However this NP completeness can be broken if one assumes tasks of equal duration in which case EDF is again optimal. Moreover we will show that EDF is optimal in non preemptive scheduling for non concrete task sets; this result was known for periodic tasks. We show that this result still hold for aperiodic task sets and we derive a necessary and sufficient condition for schedulability. This paper also reviews numerous results that have been established for preemptive scheduling, we study whether or not these results hold for non-preemptive and if they can be adapted.

This paper is divided in five sections, this plan allows to simply classify the result by distinguishing between aperiodic and periodic task sets. The last section is devoted to idling scheduling and is just a brief introduction.

Section 2 is devoted to introducing the models and the notations used

throughout this paper.

Section 3 is devoted to non idling and non preemptive scheduling of aperiodic tasks. This sections encompasses a few technical results concerning complexity and condition for optimality of the Earliest Deadline First algorithm (EDF).

Section 4 is devoted to the analysis of non idling and non preemptive scheduling of periodic tasks. One shows a few technical results concerning complexity, time interval to test feasibility, necessary feasibility condition. We show an interesting result which state that the general feasibility condition for preemptive scheduling has no equivalence in non preemptive scheduling even if one takes into account the blocking factor.

Section 5 is a very quick introduction to idling non preemptive scheduling.

2 Notations and definitions

We consider the scheduling problem of a set a()={a(1),...,a(n)} of n tasks a(i), $i \in [1, n]$ over a single processor.

By definition:

- A task is said **concrete** if its release time is known “a priori” otherwise it is **non-concrete**. Then, an infinite number of concrete task sets can be generated from a non-concrete task set.
- An **aperiodic** task is invoked once when a **periodic** (or **sporadic**) task recurs.
  Periodic and sporadic tasks differ only in the invocation time. The $(k+1)^{th}$ invocation of a periodic task occurs at time $t_{k+1} = t_k + p_i$ while it occurs at $t_{k+1} \geq t_k + p_i$ if the task is sporadic.

Notations:

- a concrete aperiodic task a(i), consists of a triple (r_i, e_i, d_i) where r_i is the absolute time the task is released, e_i the execution time and d_i the relative deadline. A concrete periodic (or sporadic) task a(i), is defined by (r_i,e_i,d_i,p_i) where p_i is the period of the task.
- a non-concrete aperiodic task a(i) consists of (e_i,d_i). A non-concrete periodic (or sporadic) task a(i) is defined by (e_i,d_i,p_i).

Throughout this paper, for the sake of simplicity, we shall use a(i) to describe the task and its parameters. For example, we shall write $a(i)=(r_i, e_i, d_i)$ for a concrete aperiodic task.

Furthermore, by definition:

- A **non-preemptive** scheduling policy does not interrupt the execution of any task.
- With **idling** scheduling policies, when a task has been released, it can either be scheduled or wait a certain time before being scheduled even if the processor is not busy.
- With **non-idling** scheduling policies, when a task has been released, it cannot wait before being scheduled if the processor is not busy. Notice that an idle period, i.e. no pending tasks in this case, can have a zero duration.
- A concrete task set a() is said to be **synchronous** if there is a time when $r_i=r_j$ for all tasks i, $j \in [1, n]$; otherwise, it is said to be **asynchronous** (the problem of deciding whether an asynchronous task set can be reduced to a synchronous one has been shown to be NP-complete in [LM80]).
- A concrete task set a() is said to be **valid (schedulable)** if it is possible to schedule the tasks of a() (including periodic recurrences in the case of periodic or sporadic task sets) so that no task ever misses a deadline when tasks are released at their specified released times.
- A non-concrete task set a() is said to be **valid (schedulable)** if every concrete task set that can be generated from a() is schedulable.
- A scheduling policy is said to be **optimal** if this policy finds a valid schedule when any exists.

Throughout this paper, we assume the following:

- the **EDF** scheduling policy uses any fixed tie breaking rule between tasks when they have the same absolute deadline (i.e. release time + relative deadline).
- NINP-EDF denotes Non Preemptive Non Idling EDF.
- time is discrete (tasks invocations occur and tasks executions begin and terminate at clock ticks; the parameters used are expressed as a multiples of clock ticks); see [BHR90] for a justification.

3 Non-idling and non-preemptive scheduling of aperiodic tasks

3.1 Concrete tasks

3.1.1. Complexity

It has already been shown that the non preemptive scheduling problem of aperiodic tasks is a NP problem [GA79]. However in the proof that of [GA79] the set of tasks which is used exhibits a load 1. In the following we show a new result; the scheduling problem of aperiodic tasks is a NP problem even if we only consider the scheduling of set of tasks exhibiting a load lower that any given number.

Theorem 3.1: The non-idling non preemptive scheduling problem of n tasks is a NP complete problem even if the task set exhibits a load lower than any given number.

We need first to define the load for a given concrete aperiodic task set: a(i)=(r_i, e_i, d_i) of n tasks.

Let us define
$t_b = min_{1 \le i \le n}\ r_i$ and t_e the end of the execution of a(i) under any arbitrary non idling scheduling policy. We define the load C as:

$$C = \frac{\sum_{i=1}^{n} e_i}{t_e - t_b}$$

Proof: Actually the idea of the proof is simple; let us consider an already introduced task set leading to a NP complete scheduling problem for instance similar to the one in [GA79]. We include this task set into a larger task set. The parameters are tuned such that the execution time of the former task set can be made as small as wished compared to the total execution time of the task set.

Let us consider the following tasks:

$$1 \le i \le m \qquad a(i) = (0, 1, \Omega)$$

$$m + 1 \le i \le 2m \qquad a(i) = (\Omega + 2(i - m) - 1, 1, \Omega + 2(i - m))$$

$$2m + 1 \le i \le 5m \qquad a(i) = (\Omega, s(i - 2m), \Omega + 2m)$$

where one has the following constraints on the s(i)

$$1 \le i \le m \qquad \frac{1}{4} < s(i) < \frac{1}{2} \qquad \sum_{k=1}^{3m} s(k) = m$$

we also assume that $\Omega \gg m$. One can easily show that the m first tasks can be almost arbitrarily be scheduled in the time interval $[0, \Omega]$. The m leading tasks have no laxity, their schedule leaves therefore m intervals of length 1 for the 3m last tasks. Therefore the feasibility of this task set is equivalent to the schedulabilty of the 3m last tasks. It is obvious that this latter problem is equivalent to the 3-PARTITION problem which is stated as: is it possible to find in the 3m last tasks, m sets of three tasks such that

$$1 \le i \le m \qquad \sum_{k \in A_i} s(k - 2m) = 1 ?$$

Meanwhile the load generated by the task set is equal to:

$$C = \frac{3m}{\Omega + 2m}$$

and can be made smaller than any given number.

End of proof.

One has seen that the scheduling of n concrete aperiodic tasks $a(i)= (r_i, e_i, d_i)$ is actually a NP complete problem. Therefore we can not expect a polynomial test, however we will show in the following that if the scheduling interval comprises p busy periods, each of them encompassing n_p tasks, the complexity of the problem will be $n_1!+ n_2!+ ....+n_{p-1}!+n_p!$. Actually this result is an obvious result, it is based on the following lemma.

Lemma 3.1: The length of the busy period does not depend on the scheduling used to execute tasks.

Proof: A busy period starts with the arrival of at least one task. At the beginning of the busy period the workload is the sum of the execution times of the arrived tasks. This workload decreases of one unit per time unit whatever be the applied schedule. At each new task arrival the duration of the incoming task is added to the workload. The workload indicates the remaining execution time to execute all the pending tasks. When the workload reaches 0, it is the end of the busy period. Therefore the duration of a busy period does not depend on the used scheduling scheme.
End of proof

Theorem 3.2: If the scheduling interval comprises p busy periods, each of them encompassing n_i tasks, the complexity of the problem will be $n_1!+ n_2!+ ....+n_{p-1}!+n_p!$

Proof: This theorem is a simple consequence of the fact that we only consider non idling scheduling. Within a busy period, we can at most schedule the tasks involved in a busy period, therefore in the busy period i we have at most $n_i!$ scheduling choice. Moreover the scheduling sequences in busy periods are of course independent, the overall scheduling problem is to schedule tasks in the successive busy periods. Therefore the complexity of the problem is at most: $n_1!+ n_2!+ ....+n_{p-1}!+n_p!$

Since the determination of the busy period does not depend on the scheduling scheme we can for instance use the simple first arrived first served rule. This technique will avoid scheduling computation.

End of proof

Let us assume that the concrete aperiodic tasks $a(i)= (r_i, e_i, d_i)$ are ordered according to the p busy periods and that the tasks are ordered such that the release times appear in a non decreasing manner: $r_i \leq r_j$ when $i < j$.

Let us consider the scheduling in a busy period i and r the release times of the last task in this busy period.

Theorem 3.3: After the instant r and until the next release time (necessarily to a task belonging to another busy period), NINP-EDF will lead to a valid schedule.

Proof: After the instant r and until the end of the busy period there is no arrival of task. Without loss of generality let us denote by x the end of the currently executed task, y the end of the busy period and b(i) $1 \leq i \leq k$ the remaining tasks to be scheduled in the busy period taking into account the schedule until time x.

We will prove that if any valid schedule completing the schedule until time x exists, NINP-EDF can schedule the remaining tasks. Let us prove this by induction on k, the number of remaining tasks. For k=1 the result is obviously true. Let us suppose that it is true for k and let us prove that is true for k+1. At time x, let us schedule the task with the shortest deadline (random choice if case of several equal deadlines). If any valid exists after time x this task must be schedulable. At the end of the task we have left k task and by induction we know that NINP-EDF will schedule the tasks if any valid schedule is available.
End of proof

3.1.2. Earliest deadline first

Since we have shown that scheduling of aperiodic tasks in non-preemptive scheduling is NP we already know that the EDF algorithm is not optimal. Below is an example of a valid schedule which is not derived from an EDF algorithm.

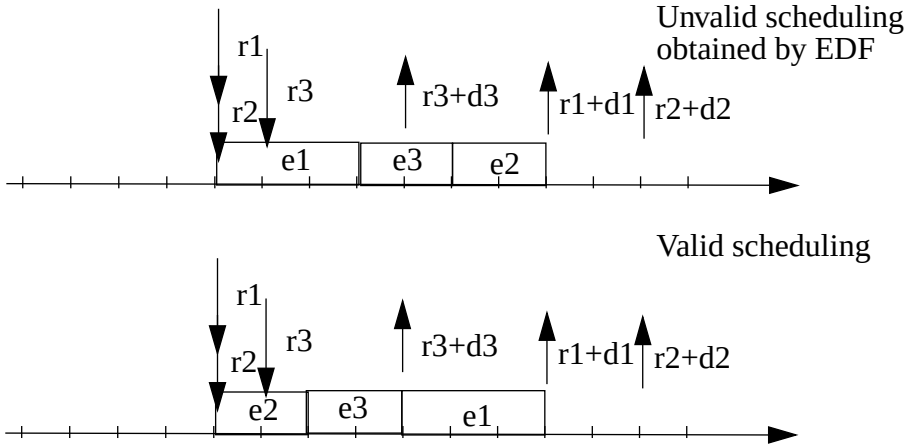

Figure 3.1: EDF is not optimal in non premptive scheduling

Actually at the arrival of task 3, the blocking effect of task 1 is such that the deadline of task 3 can not be met. However if task 2 is started first contrary to the EDF rule, task 3 can satisfy its deadline and it can be noticed in the proposed example that task 1 satisfies its deadline if even if executed before task 2 This blocking effect is as far as non preemptive scheduling is concerned a key point.

One can see that the durations of the tasks are key parameters. It is therefore not astonishing to observe that if we have constraints on the task duration this NP completeness can vanish. In the following, we deal with the case when the tasks have all the same durations. In that case, one can show that NINP-EDF is optimal. That is shown in the following theorem.

Theorem 3.2: NINP-EDF is optimal in the presence of any sequence of n concrete tasks with equal duration.

Proof: Let s() be a valid schedule; we will first show that if two tasks are not scheduled according to EDF the permutation of the execution of these two tasks lead to a valid schedule. Let us see figure 3.2 and suppose that in the initial schedule, task i is scheduled prior task j. Let us denote by t_i and t_j the beginning of the execution of these two tasks in this schedule. Since we

assume that tasks i and task j are not scheduled according EDF, we therefore have $r_j+d_j<r_i+d_i$ and the task j is released at time t_i. Let us switch the execution of tasks i and j. Since all the tasks have the same duration the starting time of all the tasks executed between task j and task i are left unchanged. Therefore the scheduling of these tasks is still valid and the obtained schedule of the task set is still valid.

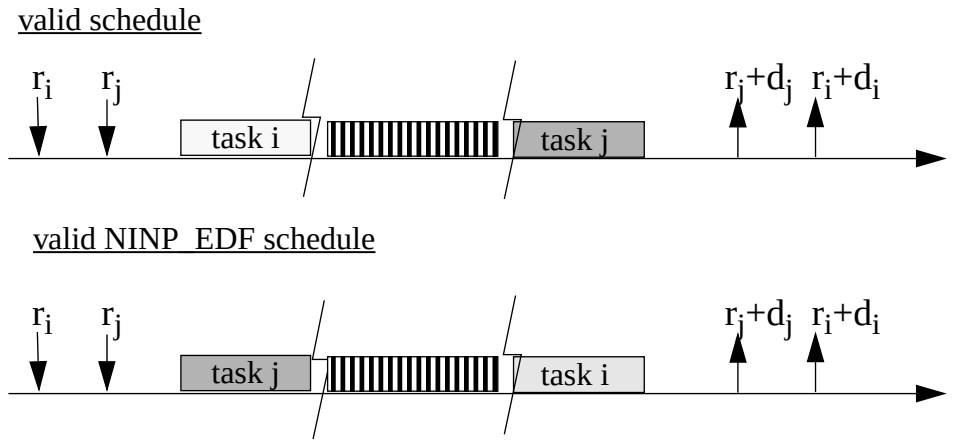

Figure 3.2: Reranking of task i and j.

One considers the following algorithm: let us select the first scheduled task and the other tasks released at the starting time of this task. Among all these tasks one selects the task with the smallest deadline (tie if equality between deadlines). If this selected task has a smallest deadline than the first scheduled task the execution of the two task are exchanged. We then select the second scheduled task and we recursively apply the same algorithm. By construction this schedule is valid. Moreover the obtained schedule is EDF since at a given time the scheduled task is the task with the smallest deadline. This completes the proof, EDF is optimal.
EndProof

3.2 Non-concrete aperiodic tasks

In the following, we consider the scheduling problem for a() a task set of n non-concrete aperiodic tasks $a(i)= (e_i, d_i)$. We will first give necessary

conditions for this task set to be feasible. We then show that this condition is sufficient and that EDF actually can schedule the task set. An obvious consequence is that EDF is optimal for non-concrete aperiodic task set.

Theorem 3.4: Let $a() = \{a(1), a(2), \ldots, a(n)\}$, where $a(i) = (e_i, d_i)$, be a set of n non-concrete aperiodic tasks sorted in increasing order by relative deadline (i.e., for any pair of tasks a(i) and a(j) such that i>j then $d_i \geq d_j$).
If a() is feasible then

(C1) $\forall i, 1 < i \leq n \;\; ; \forall j, 1 \leq j < i : \qquad d_j \geq e_i - 1 + \sum_{k=1}^{j} e_k$

Proof: For that purpose, let us define D_{T_1,T_2} the processor demand in the time interval $[T_1, T_2]$. D_{T_1,T_2} is the maximum amount of processing time required by a concrete task set b() generated from the non-concrete task set a() in the interval $[T_1, T_2]$. D_{T_1,T_2} will be a function of release time, execution time and deadline of each tasks. More precisely D_{T_1,T_2} will include:

- all tasks with deadlines in the interval $[T_1, T_2]$ (complete or remaining execution time).
- some tasks with deadlines greater than T_2 (if there are instants in the interval $[T_1, T_2]$ during which only tasks with deadlines greater than T_2 are pending).

If b() is schedulable then necessarily for every interval $[T_1, T_2]$, $D_{T_1,T_2} \leq T_2 - T_1$.

We will show that the condition is necessary by showing the contraposive. If a() does not satisfy (C1) then there exists a concrete task set b() (generated from a()) that is not schedulable.Let us assume that (C1) is not met, thus

$$\exists i, 1 < i \leq n \;\; ; \exists j, 1 \leq j < i \text{ such that } d_j < e_i - 1 + \sum_{k=1}^{j} e_k .$$

Let us consider the concrete task set b() shown in figure 3.3, generated from a(), where for some value of i, $1<i\leq n$, $r_i=0$ and where the other tasks are released at 1. Since all the tasks are released at time 1, we know from theorem 3.3 that EDF is optimal after the end of execution of task i.

We then have: $$d_j + 1 < e_i + \sum_{k=1}^{j} e_k = D_{0,(d_j+1)} .$$

Indeed $D_{0,(d_j+1)}$ consists of the cost of:

- the execution of task b(i) (since neither preemption nor inserted idle time are allowed, task b(i) must be executed in the interval [0, e_i]).
- plus the processor demand due to the tasks 1 through j in the interval [1, d_j+1]. Since (C1) does not hold and since the tasks are sorted in increasing order by deadline, tasks with relative deadlines greater than or equal to d_j do not contribute to this processor demand.

As (C1) does not hold then $d_j + 1 < D_{0,(d_j + 1)}$ and hence b() is not schedulable. This completes the proof of theorem 3.4.

End of proof

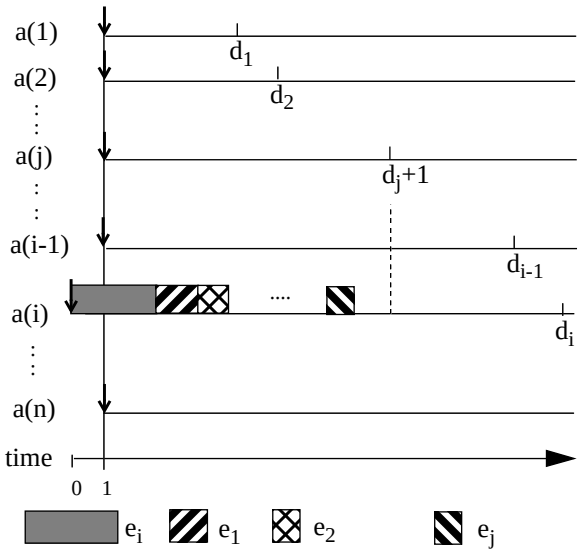

Figure 3.3:Non feasible concrete task set

Theorem 3.5: Let a() = {a(1), a(2), …, a(n)} , where a(i) = (e_i,d_i) , be a set of n non-concrete aperiodic tasks sorted in increasing order by relative deadline (i.e., for any pair of tasks a(i) and a(j) with i>j, then $d_i \geq d_j$).

If condition (C1) $\forall i, 1 < i \leq n \;\; ; \forall j, 1 \leq j < i : \;\; d_j \geq e_i - 1 + \sum_{k=1}^{j} e_k$

holds then a() is feasible. Moreover EDF can schedule a().

Proof: One will show this result by contradiction that. Let us suppose that a() satisfies (C1) but is not schedulable. In other words, there exists at least one concrete task set b() (generated from a()) such that b() is not schedulable, $\forall i, 1 \leq i \leq n \quad b(i) = (r_i,e_i,d_i)$. Let s() be a non valid schedule of b() such that the deadline of b(j) is not met at time T ($T=r_j+d_j$). Consider now T_0 the end of the last idle period before T. There are two cases during the busy period $[T_0,T]$:

- all the scheduled tasks have their absolute deadline less than or equal to T.
- the opposite; at least one of the scheduled tasks has its deadline after T.

In the first case and since T_0 is the beginning of a busy period, $D_{T_0,T}$ the processor demand during the busy period $[T_0,T]$ is such that:

$$D_{T_0,T} \leq \sum_{k=1}^{n} \delta_{(T_0 + d_k \leq T)} e_k \qquad \text{where} \quad \delta_{(T_0 + d_k \leq T)} = 1 \quad \text{if}$$

$T_0 + d_k \leq T$ and 0 else.

Indeed, $D_{T_0,T}$ does not include:

- pending tasks released before T_0 by definition of T_0,
- tasks released after (or at) T_0 with relative deadline greater than $T-T_0$ since we are in the first case (all the scheduled tasks during the busy period $[T_0,T]$ have their absolute deadline less than or equal to T).

At the same time, due to the missed deadline, $T - T_0 < D_{T_0,T}$ and therefore one obtains:

$$T - T_0 < \sum_{k=1}^{n} \delta_{(T - T_0 \geq d_k)} e_k \ .$$

Moreover as $T= r_j+d_j$ and as $r_j=T-d_j<T_0$ is impossible by definition of T_0, therefore $r_j \geq T_0$.

We have the following inequalities:

$$T - T_0 \geq d_j,$$

$$T - T_0 < \sum_{k=1}^{n} \delta_{T - T_0 \geq d_k} e_k = \sum_{k=1}^{j} e_k$$

which lead to $d_j < \sum_{k=1}^{j} e_k$.

Since (C1) implies that $\forall j, 1 \le j \le n$: $d_j \ge \sum_{k=1}^{j} e_k$: there is a clear contradiction.

In the second case (see figure 3.4), let us consider the scheduled task a(i) as being the last task which is scheduled during the busy period $[T_0,T]$ having a deadline after T. Let us denote by T_i the start time of the execution of a(i). Similarly to the first case, the processor demand $D_{T_i,T}$ during the busy period $[T_i,T]$ is such that:

$$D_{T_i,T} \le e_i + \sum_{k=1}^{n} \delta_{(T - T_i - 1 \ge d_k)} e_k$$

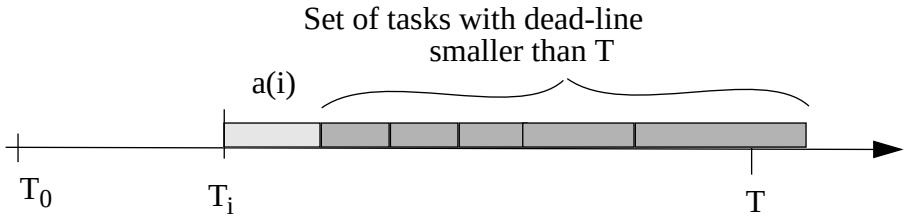

Figure 3.4: schedule with a fired deadline

Indeed, if we use NINP-EDF, $D_{T_i,T}$ does not include:

- pending tasks at T_i with relative deadline smaller than d_i (otherwise, due to NINP-EDF, they should have been executed instead of a(i)).
- pending tasks at T_i (except a(i)) with deadline greater than T. Since a(i) is the last scheduled task during $[T_0,T]$ with a deadline greater than T.
- tasks released after T_i with relative deadlines greater than $T-T_i-1$ since any scheduled task after T_i has its absolute deadline less than or equal to T.

At the same time, due to the missed deadline, $T - T_i < D_{T_i,T}$ and therefore one obtains:

(b) $T - T_i < e_i + \sum_{k=1}^{n} \delta_{(T - T_i - 1 \geq d_k)} e_k$

Moreover as $T=r_j+d_j$ and as $r_j=T-d_j < T_i+1$ is impossible (otherwise, due to NINP-EDF, a(j) would have been executed instead of a(i)), we necessarily have $r_j \geq T_i + 1$.

We have therefore the following inequalities:

$$T - T_i \geq d_j + 1,$$

$$T - T_i \leq e_i + \sum_{k=1}^{n} \delta_{((d_j = T - T_i - 1) \geq d_k)} e_k = e_i + \sum_{k=1}^{j} e_k \quad .$$

This will lead to $d_j < e_i - 1 + \sum_{k=1}^{j} e_k$ which contravenes to our initial conditions (C1).

EndProof

As any non-concrete aperiodic task set a() verifying the necessary conditions is scheduled by NINP-EDF, it follows that:

- the condition is also sufficient.
- NINP-EDF, is optimal in presence of any non-concrete aperiodic task set.

EndProof

The theorem 3.4 and 3.5 are inspired by [JE91] (which establishes the optimality of NINP-EDF and a pseudo-polynomial necessary and sufficient feasibility condition for any non-concrete periodic/sporadic task sets). Also notice that the condition (C1) is in $O(n^2)$ and that in [CHE87], a feasibility condition is given for aperiodic preemptive task set which could have been used to establish the above theorem.

4 Non-idling and non-preemptive scheduling of periodic tasks

4.1 Concrete tasks

In this section, we first show that non preemptive scheduling is NP even with a task set exhibiting a load less any given value.

We then study whether feasibility conditions established with preemptive scheduling for concrete periodic task sets can be applied or adapted in the non-preemptive context. First, we extend the result of [LM80] to non-preemptive scheduling. More precisely, we show that the necessary and sufficient conditions to determine the feasibility of a concrete periodic task sets is the feasibility on the two first period of the least common multiple of the periods. Then, we study if the results in preemptive scheduling which states that non feasibility can only the consequence of local overload still holds in non preemptive scheduling. We will show that it is, in general, not the case.

4.1.1. Complexity

The problem of knowing whether in non-idling context, a non-preemptive set of concrete periodic tasks (defined as $a(i)=(r_i,e_i,d_i,p_i)$ for i=1...n) is schedulable has been shown NP-Complete in the strong sense by [JE91]. Usually the proof that one can find are based on task sets leading to the problem of reduction towards a 3-Partition problem [JE91], these tasks sets usually exhibit a load of 1. We will show that this result still holds with a load less than any given value. Actually the proof will be based on a special task set where non preemptive and preemptive scheduling lead to the same requirements.

Theorem 4.1: The non-idling non preemptive scheduling problem of n tasks is a NP complete problem. This result holds with a task set exhibiting a load less than any given value.

Proof: Let us consider the following tasks:

$$1 \leq i \leq m \qquad a(i) = (r_i, e_i, d_i, p_i)$$

with

$$r_i = a_i, e_i = \frac{1}{f(k-1)}, d_i = 1, p_i = b_i$$

In [LM80] it is shown that the task set is feasible in a preemptive scheduling if and only if there is no k simultaneous identities of the types

$$x \equiv a_i (\mathrm{mod}\ b_i)$$

Actually the proof is similar in the case of non preemptive scheduling. Let us suppose that the task set is feasible in that case there is no k simultaneous identities of the types

$$x \equiv a_i (\mathrm{mod}\ b_i)$$

unless the task set is not feasible; a deadline will obviously be fired after time x.

If there is no k simultaneous identities of the types

$$x \equiv a_i (\mathrm{mod}\ b_i)$$

in that case we have less than k simultaneous arrival at any given time. At a time t of a task arrival, there is no pending task since the previous arrival has occurred at least k-1 time unit before and at that time we had less than k arrival. At time t according to the assumption, we have less than k arrivals and each of them will meet its deadline.

Let $((a_1,b_1),(a_2,b_2),.....,(a_i,b_i),.....,(a_n,b_n))$ be n pair of positive integers.

Is it possible to find k simultaneous identities of the type

$$x \equiv a_i (\mathrm{mod}\ b_i)$$

is shown in [BHR90] to be NP. Therefore the non preemptive scheduling of concrete periodic tasks is NP. The load of the task set can be easily expressed as

Since f can be any integer larger or equal to 1, the load of the task set can be

made smaller any given figure.

End of proof

4.1.2. Necessary and sufficient conditions

Let us introduce:

- P = least common multiple of $\{p_1,...,p_n\}$ the periods of a task set a().
- r = $\max\{r_1,...,r_n\}$ (without loss of generality, we assume that $\min\{r_1,...,r_n\} = 0$).

This part shows for non-idling, non-preemptive scheduling policy similar results than [LM80] for preemptive scheduling. The goal of this part is to show the following theorem:

Theorem 4.2: Let a() an asynchronous concrete periodic task set (defined as $a(i)=(r_i,e_i,d_i,p_i)$ for i=1...n with $0 < e_i \le d_i \le p_i$) a() is feasible on one processor if and only if

(i) $\sum_{i=1}^{n} \frac{e_i}{p_i} \le 1$,

(ii) a schedule s() exists where all deadlines in the interval [0, r + 2P] are met for all the tasks in the periodic task set.

Lemma 4.1: Let a() an asynchronous concrete periodic task set (defined as $a(i)=(r_i,e_i,d_i,p_i)$ for i=1...n with $0 < e_i \le d_i \le p_i$ with

$$\sum_{i=1}^{n} \frac{e_i}{p_i} \le 1$$

then the end of the busy period starting (or continuing) a time r+P is in the interval $[r + P, r + 2P]$. At that time denoted r+P+y the processor is idle We will denote y the shortest instant greater than r+P.

Proof: If such an instant does not exist that will mean that we have a busy processor on a time interval of length P. The system is, in that case, overloaded

and thus

$$\sum_{i=1}^{n} \frac{e_i}{p_i} > 1$$

which is in contradiction with our hypothesis. It can be noticed that this instant t can be such that at t the processor is just finishing the last pending task and a t+ we have new incoming tasks. In figure 4.1 we denote by r+P+y this instant

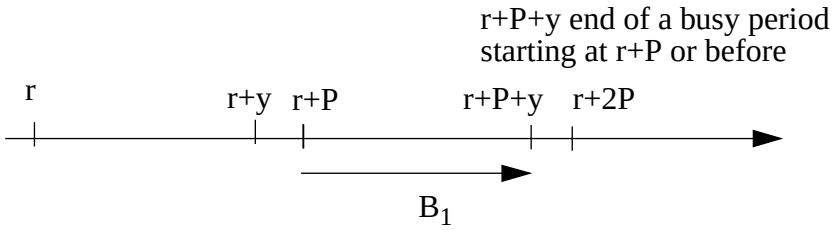

Figure 4.1: End of the busy period including r+P

End of proof

The next lemma is to show that the processor is idle at time r+y. See figure 4.2.

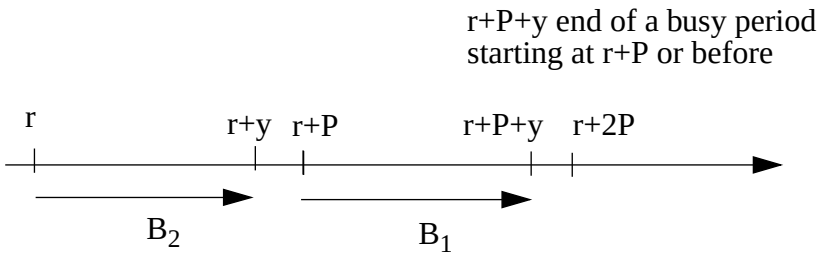

Figure 4.2: The processor is idle at r+y

Lemma 4.2: With the same hypothesis that lemma 3.1, the processor is idle at time r+y. Let us denote by W(r+y) the remaining work at time r+y (remaining workload pending or being executed at time r+y).

Proof:
Let us denote by B1 the busy period which stops at a time r+P+y and B2 the last busy period starting before r+y. Since the tasks are only all present in the

system at time r, it is possible that B2 actually encompasses less task arrivals than task arrivals in the interval B2. Therefore

$$W(r + y) \leq W(r + P + y) = 0$$

The processor is idle at time r+y.
End of proof

We can prove the theorem 4.2.

Proof: Of course the necessary condition is obvious. As a consequence of the previous lemmas, one has found y such that the processor is idle at time r+y and time r+P+y. Therefore the system is, as remaining tasks are concerned, exactly in the same situation at time r+y and r+P+y, the pattern of arrivals is also similar in the intervals $[r + y, r + P + y]$ and $[r + P + y, r + 2P + y]$ thus if we use the schedule of the interval $[r + y, r + P + y]$ we can derive a translated schedules in the intervals $[r + kP + y, r + kP + 1 + y]$, these schedules will be valid an provide a valid schedule for all $t > 0$.

End of proof

However the following example provides and example which shows that the feasibility of [0,r+P] is not sufficient to ensure the feasibility of the task set. Let us consider the following task set:a(1)=(r_1=0,e_1=4,d_1=5,p_1=10) and a(2)=(r_2=3, e_2=3,d_2=4,p_2=5) We can easily show on figure 4.3 that this task set can be scheduled on [0,r+P] but can not be scheduled on [r+P,r+2P].

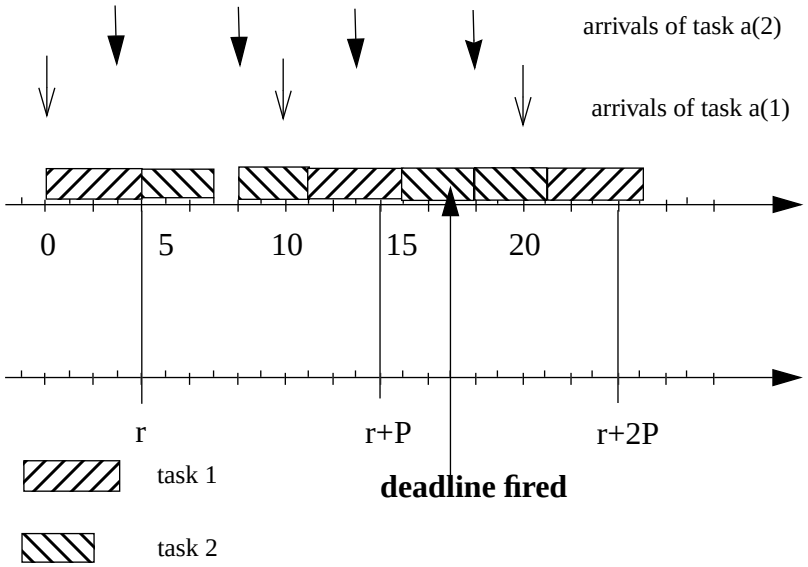

Figure 4.3: Task set feasible on [r,r+P] but not on [r+P,r+2P]

In preemptive scheduling, we have the well known and general feasibility test which states that a task set is feasible if and only if the workload on any given interval is less that the duration of the interval (no overload on any given interval). One may wonder if this result or a similar one could hold in non-preemptive scheduling. Of course, we have first to find a necessary condition. As we deal with non-preemptive scheduling one already knows that we have to take into account the blocking factor. As a matter of fact, we will be able to show the following result.

Theorem 4.3: Let a() an asynchronous concrete periodic task set (defined as $a(i)=(r_i,e_i,d_i,p_i)$ for i=1...n); if the task set a() is feasible with the schedule S then:

$$\forall(t_1, t_2) \qquad \sum_{i=1}^{n} \eta_i(t_1, t_2)e_i + R_S(t_1) \leq t_2 - t_1$$

where $R_S(t)$ denote for a given scheduling S the remaining execution time of the task currently executed at time t_1 and $\eta_i(t_1, t_2)$ denotes the number of

time that task i arrives after time t_1 with a deadline before t_2. Formally $\eta_i(t_1, t_2)$ denotes therefore the number of integer k satisfying $r_i + kp_i \geq t_1$ and $r_i + kp_i + d_i \leq t_2$.

Proof:
In a given time interval $[t_1, t_2]$ we must finish the execution of a possibly scheduled task and we must handle all the tasks arrived after t_1 and whose deadlines are before t_2 and we must finish the remaining worload $R_S(t_1)$. Therefore, we necessarily have :

$$\sum_{i=1}^{n} \eta_i(t_1, t_2)e_i + R_S(t_1) \leq t_2 - t_1$$

End of proof:

Actually one may wonder if this condition is sufficient. In other words the question is: is true that with a non feasible task set, a deadline is fired will necessarily be when in a given interval the blocking factor at t_1 plus the requested execution time in this interval exceeds the duration of the interval. Actually it is not the case for instance, we can see that on the following example:

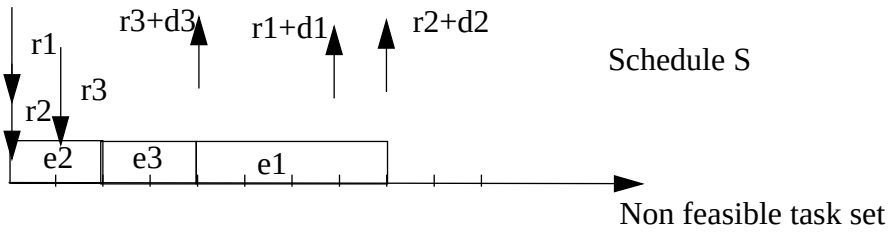

Figure 4.3: Example of a non feasible task set as the schedule S satisfies condition of theorem 4.3

in that case we have:

$$\sum_{i=1}^{3} \eta_i(0, 4)e_i = 2 \qquad R_S(0) = 0 \qquad \sum_{i=1}^{3} \eta_i(1, 4)e_i = 2 \qquad R_S(1) = 1$$

$$\sum_{i=1}^{3} \eta_i(0, 7)e_i = 6 \qquad R_S(0) = 0 \qquad \sum_{i=1}^{3} \eta_i(1, 7)e_i = 2 \qquad R_S(1) = 1$$

$$\sum_{i=1}^{3} \eta_i(0,8)e_i = 8 \qquad R_S(0) = 0 \qquad \sum_{i=1}^{3} \eta_i(1,8)e_i = 2 \qquad R_S(1) = 1$$

Therefore the given inequalities hold meanwhile S can not schedule the task set. It can be noticed that the task set is not feasible.

4.1.3. Tasks of equal duration

In the following we will give a necessary and sufficient conditions of feasibility for asynchronous concrete periodic tasks sets with the additional assumption that all the tasks have the same duration. We have the following theorem

Theorem 4.4: Let a() an asynchronous concrete periodic task set (defined as $a(i)=(r_i,e_i,d_i,p_i)$ for i=1...n) then if and only if

$$\forall(t_1, t_2) \qquad \sum_{i=1}^{n} \eta_i(t_1, t_2)e_i + R_S(t_1) \leq t_2 - t_1$$

where S is an arbitrary schedule then the concrete periodic task set is feasible.

Proof: Actually we will show that if one assumes the given inequalities then it is possible to show that NINP-EDF will schedule the task set. For that let us suppose that the given inequalities hold and that NINP-EDF can not schedule the task set. In that case we have at least one fired deadline, let us call t_2 this very deadline. See figure 4.4.

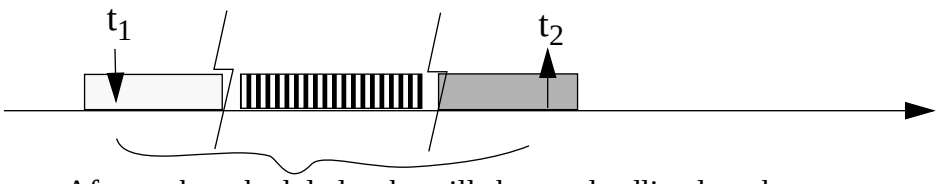

Figure 4.4: Non preemptive schedule with the same duration

Let us call t_1 the first instant such that after t_1 all the scheduled tasks have a deadline greater than t_2. At t_1-1 and before all the scheduled task will have a deadline greater than t_2. Since the used scheduling is NINP-EDF then the tasks scheduled after t_1 are necessarily arrived after t_1. Therefore since at time t_1 the remaining time for the currently executed task is for a given schedule S : $R_S(t_1)$:

$$\sum_{i=1}^{n} \eta_i(t_1, t_2)e_i + R_S(t_1) > t_2 - t_1$$

which is in contradiction with our hypothesis.
End of proof.

Actually contrary to the preemptive case, the synchronous is not a special case with the most stringent constraints it is only a given instantiation of the asynchronous case. In the figure 4.5 below is an example of non feasible synchronous task set a(1)=(0,2,8,8); a(2)=(0,3,5,8); a(3)=(0,2,3,8) when a scenario of asynchronous activation provides a non feasible task set: a(1)=(0,2,8,8); a(2)=(0,3,5,8); a(3)=(1,2,3,8).

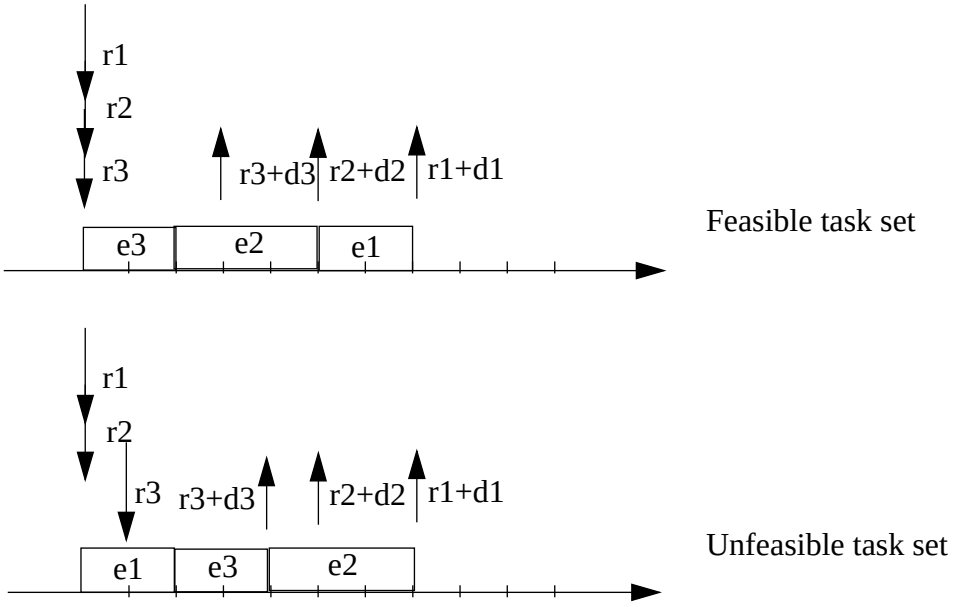

Figure 4.5: Feasible synchronous task set associated with an unfeasible asynchronous task set

4.2 Non-concrete tasks

The scheduling problem of non-concrete tasks is extensively studied in [JE91]. There are two main results in this scheduling problem. The first one is that EDF is optimal and the second one (which is a consequence of the first one) is that a necessary and sufficient condition of schedulability can be easily expressed. These two results are proved in [JE91]; we give them below without justification the necessary and sufficient condition given in [JE91].

Theorem 4.5: Let $a() = \{a(1), a(2), \ldots, a(n)\}$, where $a(i)=(e_i,d_i,p_i)$ be a set of n non-concrete aperiodic tasks sorted in increasing order by period durations (i.e., for any pair of tasks a(i) and a(j) with i>j, then $p_i \geq p_j$). One also assumes that $d_i = p_i$.
The two following conditions are a necessary and sufficient condition.

1) $$\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$$

2) $\forall i, 1 < i \leq n \;\; ; \;\; \forall L \in [p_1, p_i] \;\; : \;\; L \geq e_i + \sum_{k=1}^{i-1} \left\lfloor \frac{L-1}{p_k} \right\rfloor e_k$.

This result is revisited and generalized in [SZ94],[GRS96]. One can very briefly explained it by giving a simple interpretation. The first term of the second condition can be expressed as a blocking factor (i.e the remaining execution time of a task due to the non preemptive effect) and the second term is the processor load.

Actually the condition that we have given can be simplified as it is done in [GRS96] the necessary and sufficient condition can be then summarized by

$$\forall t > 0 \qquad t \geq b(t) + r(t) \quad .$$

where b(t) is the blocking factor and r(t) the processor load. r(t) is given by the classical following expression:

$$r(t) = \sum_{k=1}^{n} \left\lfloor \frac{t}{p_k} \right\rfloor e_k \quad .$$

b(t) can be simply evaluated since we use EDF (known to be optimal); therefore b(t) will simply be obtained by the task with the longest execution time and whose relative deadline is greater than t (otherwise this task will be taken into account in the processor load). b(t) is thus the duration minus 1 of the longest task with a deadline greater than t.

The result of 3.2 can be understood as a particular case of this result. As a matter of fact, in the condition given in 3.2

$$\forall i, 1 < i \leq n \;\; ; \forall j, 1 \leq j < i : \qquad d_j \geq e_i - 1 + \sum_{k=1}^{j} e_k$$

one can recognize the blocking factor e_i-1and the processor load.

5 Idling and non-preemptive scheduling

This section is a very short survey on idling non preemptive scheduling. The general problem of finding a feasible schedule in an idling and non-preemptive context is known to be NP-complete [GA79, annex 5].

Heuristic techniques can be used [MA84], [MOK83], [ZHAO87] to reduce the complexity. However, this reduction is achieved at the cost of obtaining a potentially sub-optimal solution. Optimal decomposition approaches can also be used [YUA91], [YUA94], [PC92] to reduce the complexity by dividing the n tasks into m subsets. Decomposition, however, is not possible for any task sets

In the following we will give a few examples to show differences between non idling and idling scheduling.

For example, NINP-EDF is not optimal for idling scheduling otherwise this would have contradicted the NP-completeness, e.g. the following task set (see figure 5.1) is feasible but NINP-EDF does not produce a valid schedule. Moreover this task set gives an example of a case where a valid idling scheduling exists but no non-idling scheduling can be found.

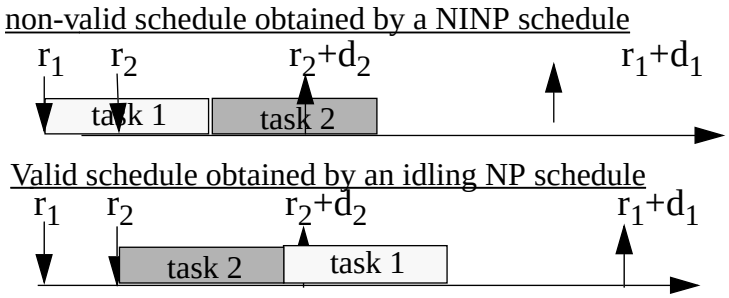

Figure 5.1: Idling scheduling may find valid schedule not found by NINP schedules

The example on figure 5.1 could lead to the following idea: an optimal (of course off-line) scheduling strategy could be such that at each instant the task with the shortest deadline is scheduled, such strategy could be used rule the idling periods of the schedule.

Unfortunately, figure 5.2 provides an example for which this strategy does not work well. These two examples give an idea of the problem complexity of

idling scheduling. The authors are currently working on the subject, for instance they conjecture that the complexity reduction which appears either for tasks with equal duration or with non-concrete task set is probably no longer valid.

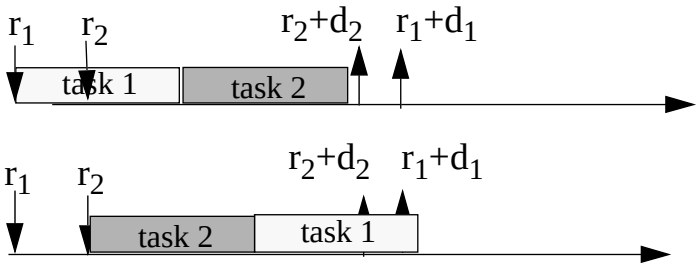

Figure 5.2: At each time the deadline with the shortest deadline must be the executed task is not a universal rule

6 Conclusion

Non preemptive scheduling is extensively studied in this paper.

In the case of non-idling scheduling of aperiodic tasks, one revisits the following result: the non preemptive scheduling problem is NP for concrete tasks. We show that the NP completeness remains true even if the workload is smaller than any given value. We also show that if all the tasks have the same duration the problem is no longer NP and can be solved by NINP EDF. We also give necessary and sufficient scheduling conditions for non idling scheduling of non concrete aperiodic tasks.

Concerning the scheduling problem of periodic tasks, one shows that the problem is still NP even with an arbitrary small load. However, we show that the schedulability can be decided on a period of twice the smallest common multiple of the tasks periods.

For non-idling scheduling, we derive a necessary schedulability condition inspired from the necessary and sufficient condition of preemptive scheduling. This condition is actually not sufficient. We show that the natural belief according to which if a task set can not be scheduled by a schedule S

then, on a given time interval, the blocking factor induced by S plus the interval worload exceeds the interval duration, is not true.

On the last section we very briefly review scheduling problems in idling non preemptive scheduling. We show that this problem is even more complex that non idling non preemptive scheduling.

7 references

[BHR90] K.Sanjoy, L.E.Rosier, R.R.Howell, “Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor”, Real Time Systems 90 p301-324.

[CHE87] H.Chetto, M.Chetto, "How to insure feasibility in a distributed system for real-time control?", Int. Symp. on High Performance Computer Systems, Paris, Dec. 1987.

[CCB91] H.Chetto, M.Chetto, T Bouchentouf. “Dynamic Scheduling of Real-Time Tasks under Precedence Constraints. The Journal of Real Time Systems, 2 , pp 181-194.

[GA79] M. R. Garey, D. S. Johnson, “Computer and Intractability, a Guide to the Theory of NP-Completeness”, W. H. Freeman Company, San Francisco, 1979.

[GRS96] Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling for general task sets. Laurent George, Nicolas.Rivierre, Marco Spuri. INRIA Research Report n^{o}2966. September 1996.

[JE91] K. Jeffay, D. F. Stanat, C. U. Martel, “On Non-Preemptive Scheduling of Periodic and Sporadic Tasks”, IEEE Real-Time Systems Symposium, San-Antonio, December 4-6, 1991, pp 129-139.

[HOW95] On Non-preemptive Scheduling Of Recurring Tasks Using Inserted Idle Times. Rodney R. Howell and Muralidhar K. Venkatrao. Information and Computation 117 pp50-62. 1995.

[KIM80] Kim, Naghibdadeh, Proc. of Perf. 1980, Assoc. Comp. Mach. pp267-276, “Prevention of task overruns in real-time non-preemptive multiprogramming systems”

[LEU82] J.Y. T. Leung and J. Whitehead, “On the complexity of Fixed-Priority Scheduling of Periodic, Real Time Tasks”, Performance Evaluation (Netherlands) 2(4), pp. 237-250(December 1982)

[LILA73] C.L Lui, James W. Layland, “Scheduling Algorithms for multiprogramming in a Hard Real Time Environment”, Journal of the Association for Computing Machinery, Vol. 20, n^{o}1, Janv 1973.

[LM80] J. Y.T.Leung, M.L.Merril, “A note on preemptive scheduling of periodic, Real Time Tasks”, Information processing Letters, Vol. 11, n^{o} 3, Nov 1980.

[MA84] P. R. Ma, "A model to solve Timing-Critical Application Problems in Distributed Computing Systems”, IEEE Computer, Vol. 17, pp. 62-68, Jan. 1984.

[MOK83] A.K. Mok, “Fundamental Design Problems for the Hard Real-Time Environments”, May 1983, MIT Ph.D. Dissertation.

[MC92] P. Muhlethaler, K. Chen, “Generalized Scheduling on a Single Machine in Real-Time Systems based on Time Value Functions”, 11th IFAC Workshop on Distributed, Computer Control Systems (DCCS’92), Beijing, August 1992

[SZ94] Shin, K. G. Zheng. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet Switched Networks. IEEE Transactions on Communications, 42-44.

[YUA91] Xiaoping Yuan, “A decomposition approach to Non-Preemptive Scheduling on a single ressource", Ph.D. thesis, University of Maryland, College Park, MD 20742.

[YUA94] Xiaoping Yuan, Manas C. Saksena, Ashok K. Agrawala, “A decomposition approach to Non-Preemptive Real-Time Scheduling”, Real-Time Systems, 6, 7-35 (1994).

[ZHAO87] W. Zhao, K. Ramamritham, J. A. Stankovic. “Scheduling Task with Resource requirements in a Hard Real-Time System”, IEEE Trans. on Soft. Eng., Vol. SE-13, No. 5, pp. 564-577, May 1987.