INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling

Laurent George, Nicolas Rivierre, Marco Spuri

N° 2966

Septembre 1996

THEME 1

Réseaux et systèmes

ISSN 0249-6399

Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling

Laurent George, Nicolas Rivierre, Marco Spuri
{Laurent.George, Nicolas.Rivierre, Marco.Spuri}@inria.fr

Thème 1 - Réseaux et systèmes

Projet Reflecs

Rapport de recherche n°2966 - September 1996

51 pages

Abstract: Scheduling theory, as it applies to hard-real-time environment, has been widely studied in the last twenty years and it might be unclear to make it out within the plethora of results available. Our goal is first to collect in a single paper the results known for uniprocessor, non-idling, preemptive/non-preemptive, fixed/dynamic priority driven contexts, considering general task sets as a central figure for the description of possible processor loads. Second to establish new results when needed. In particular, optimality, feasibility conditions and worst-case response times are examined largely by utilizing the concepts of workload, processor demand and busy period. Some classic extensions such as jitter, resource sharing are also considered.
Although this work is not oriented toward a formal comparison of these results, it appears that preemptive and non-preemptive scheduling are closely related and that the analysis of fixed versus dynamic scheduling might be unified according to the concept of higher priority busy period. In particular, we introduce the notion of deadline-d busy period for EDF schedules, that we conjecture to be an interesting parallel of the level-i busy period, a concept already used in the analysis of fixed priority driven scheduling.

Key-words: dynamic priorities, EDF, feasibility, fixed priorities, non-idling, non-preemptive, optimality, preemptive, real-time, worst-case response times, scheduling.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 LE CHESNAY Cedex (France)
Téléphone : (33) 01 39 63 55 11 - Télécopie : (33) 01 39 63 53 30

Ordonnancement monoprocesseur temps réel préemptif et non préemptif

Résumé : La théorie de l'ordonnancement, comme elle s'applique en environnement temps réel, a été largement étudiée durant les vingt dernières années et il peut apparaître difficile de s'y retrouver face à la pléthore de résultats existants. Notre objectif est premièrement de réunir ces résultats en contexte centralisé, non-oisif, préemptif/non-préemptif, à priorité fixe/ dynamique. Pour cela nous considérerons des jeux de tâches génériques afin de prendre en compte le plus de scénarios d'arrivées possibles. Deuxièmement de nouveaux résultats sont établis si nécessaire. En particulier, l'optimalité des politiques d'ordonnancement, les conditions de faisabilité associées ainsi que l'analyse des pires temps de réponse sont examinées grâce aux concepts de charge, de demande processeur et de période occupée. Des extensions classiques telles que la gigue sur les inter-arrivées ou la présence de ressources partagées sont aussi examinées.
Quoique ce travail ne soit pas orienté vers une comparaison formelle des résultats, il apparaît cependant que l'ordonnancement préemptif et non-préemptif sont très proches. De plus, l'analyse de l'ordonnancement à priorité fixe et dynamique peut être unifiée par l'utilisation des concepts de période occupée relative à une priorité (et donc dépendante de l'ordonnanceur utilisé). En particulier, nous introduisons le concept de "deadline-*d* busy period" pour l'ordonnanceur EDF qui nous parait être un point de départ intéressant pour une comparaison avec les "level-*i* busy period" utilisées dans le cas des ordonnanceurs à priorité fixes.

Mots-clé : Faisabilité, EDF, priorités dynamiques, priorités fixes, non-oisif, non-préemptif, optimalité, ordonnancement, préemptif, pire temps de réponse, temps réel.

Contents

1.	**Introduction**	**4**
2.	**Model, concepts and notations used in this paper**	**4**
2.1	Model	4
2.2	Classic concepts	5
2.3	Goals	7
3.	**Preemptive schedulers**	**8**
3.1	Dynamic priority driven schedulers	8
3.1.1	Optimality	8
3.1.2	Feasibility Condition	8
3.1.3	Worst-Case Response times	13
3.2	Fixed priority driven schedulers	15
3.2.1	Optimality	15
3.2.2	Feasibility condition and worst-case response times	16
4.	**Non-preemptive schedulers**	**17**
4.1	Idling/non-idling for non-preemptive scheduling	17
4.2	Dynamic priority driven schedulers	18
4.2.1	Optimality	18
4.2.2	Feasibility	19
4.2.3	Worst-case response times	23
4.3	Fixed priority driven schedulers	27
4.3.1	Feasibility Condition and Worst-Case Response times	27
4.3.2	Optimality	30
5.	**Shared resources and release jitter**	**34**
5.1	Dynamic priority driven schedulers	34
5.1.1	Preemptive case	34
5.1.2	Non-preemptive case	35
5.2	Fixed priority driven schedulers	35
5.2.1	Preemptive case	35
5.2.2	Non-preemptive case	36
6.	**Synthesis**	**36**
7.	**Conclusion**	**37**
References		**38**
Annexe A. Busy Periods, Definitions and Properties		**40**
Annexe B. Polynomial sufficient conditions (SC)		**47**

1. Introduction

Scheduling theory, as it applies to hard-real-time environment, has been widely studied in the last twenty years. A lot of results have been achieved for the problem of non-idling scheduling over a single processor. Although the most effective real-time schedulers make use of the HPF (Highest Priority First) on-line policy, the community of real-time researchers is currently split into two camps, those who support fixed priority driven schedulers, that were devised for easy implementation, and those who support dynamic priority driven schedulers that were considered better theoretically. Moreover, non-preemptive schedulers have received less attention than their preemptive counter parts. Therefore, on the whole, given a real-time problem, it might be unclear for non specialists to pick a solution using a theoretical analysis from the plethora of results available. The aim of this paper, considering general task sets as a central figure for the description of possible processor loads, is to:

- first focus on a generalization of the existing results regarding optimality, feasibility conditions and worst-case response times in a preemptive context. This will be done in the presence of fixed/dynamic priority driven schedulers and for several kinds of task sets;
- second extend these results, or establish new results when needed, in a non-idling, non-preemptive context.

A synoptic will be established and, while a lot of results are taken from the state of the art, some of them will be either new results or improvements on existing results. In particular, we will introduce the new concept of deadline-*d* busy period for dynamic priority driven scheduling that we conjecture to be an interesting starting point for a comparison with the level-*i* busy period used in fixed priority driven scheduling. Furthermore, it will be shown that preemptive and non-preemptive scheduling are closely related, for example in terms of optimality and feasibility properties.

The paper is organized as follows. Section 2 outlines the computational model, the concepts used and the goals. Section 3 (resp. Section 4) focuses on preemptive (resp. non-preemptive) scheduling algorithms. Some classic extensions such as jitter and resource sharing are considered in Section 5 and a synthesis is given in Section 6. Finally, Annex A focuses on the concept of busy period and Annex B examines several approaches that will allow us to establish very simple sufficient (but not necessary) conditions for general task sets.

2. Model, concepts and notations used in this paper

2.1 Model

In this paper, we shall consider the problem of scheduling a set $\tau = \{\tau_1, \ldots, \tau_n\}$ of n non-concrete periodic or sporadic tasks on a single processor. This will be done in presence of hard real-time constraints and with deadlines not necessarily related to the respective periods of the tasks. A task is a sequential job that is invoked with some maximum frequency and result in a single execution of the job at a time, handled by a given scheduler. From the scheduling point of view, a task can then be seen as an infinite number of **instances**. By definition, we consider that [JSM91]:

- A **non-concrete periodic task** τ_i recurs and is represented by the tuple (C_i, D_i, T_i), where C_i, D_i and T_i respectively represent the computation time, relative deadline and period (note that the absolute deadline of a given instance is equal to the release time of this instance plus the relative deadline). A **concrete periodic** task ω_i is defined by $\{\tau_i, s_i\}$ where s_i, the start time, is defined as the time lapse between time zero and the first instance of the task.

- A **non-concrete periodic task set** $\tau = \{\tau_1, \ldots, \tau_n\}$ is a set of n non-concrete tasks. A **concrete periodic task set** $\omega = \{\omega_1, \ldots, \omega_n\}$ is a set of n concrete tasks. Then an infinite number of concrete task sets can be generated from a non-concrete task set (without loss of generality we assume $min_{1 \le i \le n}\{s_i\} = 0$).

  A concrete periodic task set ω is called a **synchronous** task set if $s_i = s_j$ for all $1 \le i, j \le n$ (s_i is then called a **critical instant** in the literature) otherwise, ω is called an **asynchronous** task set ([LM80] showed that the problem to know if an asynchronous task set can lead to a critical instant is NP-complete).

- **Sporadic** tasks were formally introduced in [MOK83] (although already used in some papers, e.g., [KN80]) and differ only from periodic tasks in the invocation time: the $(k+1)^{th}$ instance of a periodic task occurs at time $t_{k+1} = t_k + T_i$, while it occurs at $t_{k+1} \ge t_k + T_i$ if the task is sporadic. Hence T_i represents the minimum interarrival time between two successive invocations.

- A **general** task set $\tau = \{\tau_1, \ldots, \tau_n\}$ is a non-concrete periodic or sporadic task set such as $\forall i \in [1, n]$, T_i and D_i are not related.

Throughout this paper, we assume the following:

- all the studied schedulers make use of the HPF (Highest Priority First) on-line algorithm but differ by their priority assignment scheme (in the sequel, we will no longer refer to HPF). They all make use of a fixed tie breaking rule between tasks that show the same priority. They are **non idling** (i.e., the processor cannot be inactive in presence of pending instances) and are either **preemptive** or **non-preemptive** (preemptive means that the processing of any task can be interrupted by a higher priority task).

- $\forall i \in [1, n]$, $C_i \le T_i$, $C_i \le D_i$.

- all tasks in the system are independent of one another and are critical (hard-real-time), i.e., all of them have to meet their absolute deadline. Unless otherwise stated, tasks do not have resource constraints and the overhead due to context switching, scheduling... is considered to be included in the execution time of the tasks.

- time is discrete (task invocations occur and tasks executions begin and terminate at clock ticks; the parameters used are expressed as multiples of the clock tick); in [BHR90], it is shown that there is no loss of generality with respect to feasibility results by restricting the schedules to be discrete, once the task parameters are assumed to be integers.

2.2 Classic concepts

Let us now recall some classic concepts used in hard-real-time scheduling:

- the scheduling of a concrete task set ω is said to be **valid** if and only if no task instance misses its absolute deadline.

- a concrete task set ω is said to be **feasible** with respect to a given class of schedulers if and only if there is at least one valid schedule that can be obtained by a scheduler of this class (in this paper, we consider only four classes of schedulers, by combining non-idling, preemptive/ non-preemptive, fixed/dynamic priority driven schedulers). Similarly, a non-concrete task set τ is said to be feasible with respect to a given class of schedulers if and only if every concrete task set ω that can be generated from τ is feasible in this class.
  Note that from the real-time specification point of view, a non-concrete task set is more realistic than a concrete one, since not one but all the patterns of arrival are indeed considered. For the same reasons, but from the scheduling point of view, a feasibility condition for a non-concrete task set is generally more selective and less complex than one for a concrete task set. [BHR90], [BMR90] showed that a concrete task set cannot, in general, be tested efficiently unless P=NP. Anyhow, they also showed that a concrete synchronous task set, or a non-concrete task set, can be tested in pseudo polynomial time whenever $U \leq c < 1$ (i.e., with c a constant smaller than *1*).

- a scheduler is said to be **optimal** with respect to a given class of schedulers if and only if it generates a valid schedule for any feasible task set in this class.

- $\boldsymbol{U} = \sum_{i=1}^{n} C_i/T_i$ is the **processor utilization** factor, i.e., the fraction of processor time spent in the execution of the task set [LL73]. An obvious Necessary Condition for the feasibility of any task set is that $U \leq 1$ (this is assumed in the sequel).

- $\boldsymbol{P} = lcm_{1 \leq i \leq n}\{T_i\}$ is the **base period** (or **Hyperperiod**), i.e., a cycle such as the pattern of arrival of a periodic task set recurs similarly [LM80]. Note that, even in the case of limited task sets, P can be large when the periods are prime.

- given a critical instant at time 0 ($\forall i \in [1, n]$, $s_i = 0$), the **processor demand** $\boldsymbol{h(t)}$ is the amount of computation time requested by all instances whose release times and absolute deadlines are in the interval $[0, t]$ [BMR90],[SPU96]:

$$h(t) = \sum_{i=1}^{n} max\left(0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i = \sum_{D_i \leq t} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i$$

- given a critical instant at time 0 ($\forall$i $\in$ [1, n], $s_i = 0$), the **workload** $\boldsymbol{W(t)}$ (resp. $\overline{\boldsymbol{W}}(\boldsymbol{t})$) is the amount of processing time requested by all instances whose release times are in the interval $[0, t)$ (resp. $[0, t]$) [BMR90]:

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i,$$

$$\overline{W}(t) = \sum_{i=1}^{n} \left(1 + \left\lfloor \frac{t}{T_i} \right\rfloor\right) C_i.$$

- Given a non-concrete task set, the **synchronous processor busy period** is defined as the time interval $[0, L)$ delimited by two distinct processor idle periods, i.e., any periods such that no outstanding computation exists, in the schedule of the corresponding synchronous concrete task set [KLS93]. It turns out that the value of L does not depend on the scheduling algorithm, as far as it is non-idling, but only on the task arrival pattern. In Annex A the concept of

processor busy period is formally introduced. Specific definitions of **priority busy periods** are also introduced, which are useful in the analysis of the different scheduler classes (either dynamic or fixed priority driven), and few properties are established.

- Given a non-concrete general task τ_i, $\forall i \in [1, n]$, r_i is the **worst-case response time** of τ_i, i.e., the longest time ever taken by any instance of the task from its release time until the time it completes its required computation [JP86]. Note that:

  - as shown in the sequel, the value of r_i is scheduling algorithm dependent and can be computed using the concept of priority busy period.

  - if a task has a worst-case response time greater than its period then there is the possibility for a task to re-arrive before the previous instances have completed. In our model, we consider that the new arrival is delayed from being executing after the previous instances terminate. In other words we keep the order of the events of the same task. Other ways to deal with such a situation exist (e.g. to deliver the most recent instance of the same task first), leading to an adaptation of the proposed results.

2.3 Goals

Given a scheduling context, the main goal of the existing works is to couple an "optimal" scheduler with a polynomial (or even a pseudo-polynomial-time) Feasibility Condition (**FC**) that could be either a Necessary and Sufficient Condition (**NSC**) or a Sufficient Condition (**SC**) only. A lot of results regarding optimality, feasibility conditions and worst-case response times computation are now available (see Table 1).

TABLE 1. Main existing results

	Preemptive scheduling		Non-preemptive scheduling	
	Dynamic priorities Section 3.1	Fixed priorities Section 3.2	Dynamic priorities Section 4.2	Fixed priorities Section 4.3
Optimality	[DER74], [MOK83]	[LL73], [LW82], [AUD91]	[KN80], [JSM91], [GMR95]	
Feasibility conditions	[LL73], [LM80], [BMR90], [BHR93],[KLS93], [ZS94], [RCM96]	[LL73], [JP86], [LEH90]	[KN80], [JSM91], [ZS94]	[THW94], [TBW95]
Worst-case Response times	[SPU96]	[JP86], [TBW94]		[THW94], [TBW95]

Table 1 is not an exhaustive list of papers in the field of real-time scheduling but more a summary of the main results in our knowledge. In the sequel, we will focus on these results, considering general task sets as a central figure for the description of possible processor loads, in centralized, non-idling, fixed/dynamic, preemptive/non-preemptive contexts. A synoptic (see table 2) of existing and new results for general task sets will be given in Section 6. More precisely, this paper:

- recalls existing results straight from the state of the art when a cell of the table is white.

- extends existing results from the state of the art in order to deal with general task sets when a cell of the table is coloured in grey.

- establishes new results in order to deal with general task sets when a cell of the table is coloured in grey and has no reference.

The literature regarding preemptive algorithms is quite important. Thus, Section 3 is more a description of the state of the art of the existing results such as optimal fixed/dynamic priority assignment schemes, and the associated off-line feasibility conditions and worst-case response times computation. However, for the dynamic case, we will introduce the new concept of deadline-*d* busy period in order to improve the efficiency of the NSC's. We consider this concept as a parallel with the existing concept of level-*i* busy period used in fixed priority driven scheduling (see Annex A for a formal definition).

In non-preemptive context, the literature is not so important. After a short discussion on the limited interest of idling schedulers for non-concrete task sets, Section 4 mainly recalls the existing results and establish new results when needed. It will appear that preemptive and non-preemptive scheduling are closely related, foe example in terms of optimality and feasibility properties.

3. Preemptive schedulers

3.1 Dynamic priority driven schedulers

In this section, we first recall the existing results of preemptive dynamic priority scheduling (see section 3.1.1 for the optimal scheduling algorithm, section 3.1.2 for the feasibility of a given task set and section 3.1.3 for the worst-case response times). These principles were derived in particular from [LL73], [LM80], [BMR90], [BHR93], [KLS93], [ZS94], [SPU96] and [RCM96]. At the end of the section, we will show that it is possible to combine the existing results with the new concept of deadline-*d* busy period in order to optimize the NSC.

3.1.1 Optimality

We will focus on **EDF** (Earliest Deadline First) scheduling algorithm. At any time, EDF schedules among those tasks that have been released and not yet fully serviced (pending tasks), one whose absolute deadline is earliest. If no task is pending, the processor is idle.

Theorem 1 - *([DER74]) EDF is optimal.*

The proof shows that it is always possible to transform a valid schedule into one which follows EDF (if at any time the processor executes some task other than the one which has the earliest absolute deadline, then it is possible to interchange the order of execution of these two tasks).
This optimality property is general following that any feasible task set is schedulable by EDF. Note also that the LLF (Least Laxity First) scheduler, which at any time schedules the task instance with the smallest laxity (absolute deadline minus current time minus remaining execution time), has also been shown to be optimal in the same context [MOK83], but leads to more preemptions than EDF.

Owing to its optimality property, all the results described in the sequel for dynamic priority driven schedulers are referred to EDF.

3.1.2 Feasibility Condition

■ **Case** $T_i = D_i, \forall i \in [1, n]$

Theorem 2 - *([LL73], [COF76]) For a given synchronous periodic task set (with* $D_i = T_i$, $\forall i \in [1, n]$*), the EDF schedule is feasible if and only if* $U \leq 1$.

This result gives us a simple $O(n)$ procedure based on the processor utilization to check the feasibility. The proof shows that if a given processor busy period (see Annex A) leads to an overflow at time t, the

resulting synchronous processor busy period cannot lead to an idle time prior to time t. This leads to a contradiction if we assume $U \leq 1$ and an overflow at time t.

Theorem 3 - *([LL73], [KLS93]) Any non-concrete periodic task set (with $D_i = T_i$, $\forall i \in [1, n]$) scheduled by EDF, is feasible if and only if no absolute deadline is missed during the synchronous busy period.*

This result is also valid for other sorts of task sets. Ripoll et al. [RCM96] and Spuri [SPU95, SPU96] showed that in order to study the feasibility of a non-concrete task set, in preemptive context, we can limit our attention to the synchronous processor busy period (see Annex A for a definition). All we have to do is to check the absolute deadlines in this interval. Note that [KLS93] and [SPU96] show that it is possible to consider the scheduler implementation costs in this synchronous busy period.

In the sequel, we will see that we can improve the previous results using the new concept of deadline-*d* busy period.

■ **Case** $D_i \leq T_i$, $\forall i \in [1, n]$

Historically, an NSC for this model has been known since the publication of [LM80], in which Leung and Merril established that the feasibility of an asynchronous task set can be checked by examining the EDF schedule in the time interval $[0, 2P + max\{s_i\}]$, with P the base period as defined in section 2.2; and by verifying that (1) all absolute deadlines are met during this interval and (2) the configurations (i.e. the amount of time for which each task has executed since its last instance) of the schedule at the time instants $P + max\{s_i\}$ and $2P + max\{s_i\}$ coincide (that is, the schedule becomes cyclical after the first "unstable" base period). The result was later improved by Baruah et al. [BHR90], who showed that condition (2) is always satisfied whenever $U \leq 1$.

Note that this NSC operates in exponential time in the worst-case (which is usually unacceptable) since P is in the worst case a function of the product of the task periods. On the other hand, it turns out that for a synchronous periodic task set we can restrict our attention to the interval $[0, P]$, with 0 a critical instant. That is, the synchronous arrival pattern is the most demanding for non-concrete task set.

In addition to this, the approach we are going to show now is based on the evaluation of the processor demand $h(t)$ on limited intervals such as the synchronous processor busy period (see Annex A). Indeed, being the processor demand, the amount of computation requested by all instances in the interval $[0, t]$, it follows that for any $t \geq 0$, $h(t)$ must not be greater than t in order to have a valid schedule. Note that this approach leads to a pseudo-polynomial NSC if $U \leq c < 1$.

Theorem 4 - *([BRH90], [BMR90], [ZS94], [RCM96]) A non-concrete periodic, or sporadic, task set (with $D_i \leq T_i$ $\forall i \in [1, n]$ and $U \leq 1$) is feasible, using EDF, if and only if:*

$$\forall t \in S,\ h(t) \leq t \tag{1}$$

$$\text{where } S = \left(\bigcup_{i=1}^{n} \{kT_i + D_i, k \in \aleph\} \right) \cap \left[0, min\left\{ L,\ \frac{\sum_{i=1}^{n} (1 - D_i/T_i) C_i}{1 - U} \right\} \right),$$

and L is the length of the synchronous processor busy period (see Annex A for an explanation).

A further improvement on the result of Leung and Merril [LM80] for synchronous periodic task sets is found in [BHR90] and [BMR90], where Baruah et al. show that if $U < 1$ an NSC for the feasibility of a task set is that $h(t) \le t$, $\forall t$ in the limited interval: $\left[0, \frac{U}{1-U} max_{i=1\ldots n}(T_i - D_i)\right)$.

The proof is based on the fact that if the task set is not feasible, then there is an instant of time t such that $h(t) > t$. By algebraic manipulations of this condition, an upper bound on the value of t can be determined. Recently, by using a similar manipulation on the same condition [RCM96] obtained a tighter upper bound:

$$t < h(t) \le \sum_{i=1}^{n} \left(\frac{t + T_i - D_i}{T_i} \right) C_i \le tU + \sum_{i=1}^{n} \left(1 - \frac{D_i}{T_i} \right) C_i,$$

from which:

$$t < \frac{\sum_{i=1}^{n} (1 - D_i/T_i) C_i}{1 - U}.$$

Note that if $U \le c$, with c a constant smaller than 1, this result leads to a pseudo-polynomial-time NSC [BMR90]. Note also that the interval to be checked can be large if c is close to 1.

A second upper bound on the length of the interval to be checked is given in [SPU95, RCM96]. In particular, Ripoll et al. extend to $D_i \le T_i$, $\forall i \in [1, n]$ the result of Theorem 3, by showing that the synchronous processor busy period is the most demanding one (using the same overflow argument as in [LL73]). That is, if an absolute deadline is missed in the schedule, then one is missed in the synchronous processor busy period. The computation of the synchronous busy period length L is described in Annex A.

Finally, the evaluation of the NSC is further improved as proposed by Zheng and Shin [ZS94], who propose to evaluate the processor demand only on the set S of points corresponding to absolute deadlines of task requests (i.e. the set of points where the value of $h(t)$ changes).

■ **Case** $D_i \ge T_i$, $\forall i \in [1, n]$

Theorem 5 - *([BMR90]) A non-concrete periodic, or sporadic, task set (with* $D_i \ge T_i$ $\forall i \in [1, n]$ *) is feasible, using EDF, if and only if* $U \le 1$.

The proof of [BMR90] simply shows that if $D_i \ge T_i$, $\forall i \in [1, n]$ and $U \le 1$:

$$\forall t \ge 0, \qquad h(t) = \sum_{D_i \le t} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i \le t \sum_{i=1}^{n} \frac{C_i}{T_i} \le t.$$

■ **General task sets** ($\forall i \in [1, n]$, T_i and D_i are not related)

The results seen for task sets with relative deadlines smaller than or equal to the respective periods, are also valid for general task sets, in which there is no a priori relation between relative deadlines and periods.

In particular, Theorem 3, originally conceived for task sets with relative deadlines equal to periods, was independently shown to hold also for the less restrictive models by Ripoll et al. [RCM96] and Spuri [SPU95, SPU96], respectively. That is, in order to check the feasibility of a general task set, we can still limit our attention to the synchronous busy period.

On the other hand [BMR90], [BHR93] and [ZS94] showed by algebraic manipulations of $h(t)$, as for Theorem 4, the possibility of checking the feasibility of a general task set on the following set S of points in a limited interval. The only difference with this theorem comes from the lower bound $max(D_i)$ that is needed to take into account of relative deadline greater than periods. Finally:

$$S = \left(\bigcup_{i=1}^{n} \{kT_i + D_i, k \in \aleph\} \right) \cap \left[0, max\left\{ max_{i=1\ldots n}(D_i), \frac{\sum_{i=1}^{n}(1 - D_i/T_i)C_i}{1-U} \right\} \right).$$

As [RCM96], in the case where $D_i \leq T_i \quad \forall i \in [1, n]$, it is now possible to integrate these upper bounds in one theorem for general task sets. However, if we introduce a more specialized notion of busy period, namely the *deadline busy period* (see Annex A for a formal definition), we can refine our analysis by further restricting the interval of interest. Specifically, a deadline-d busy period is a busy period in which only instances with absolute deadline before or at d are executed. It turns out that only synchronous deadline busy periods are the busy periods required in order to check the feasibility of task set.

Lemma 1 - *Given a general task set, if there is an overflow for a certain arrival pattern, then there is an overflow in a synchronous deadline busy period.*

Proof. Assume there is a pattern which causes an instance of task τ_i to miss its absolute deadline at time t. Let t' be the last time before t, such that there are no instances pending with arrival time earlier than t' and absolute deadline before or at t. By choice, t' must be the arrival time of a task instance, and there is no idle time between t' and t. Furthermore, only instances with absolute deadline before or at t are executed in $[t', t]$. Thus, t' is the beginning of a deadline-t busy period, whose length is greater than $t - t'$, by the hypothesis of overflow at time t.

Consider now the scenario in which all tasks but τ_i are released synchronously and at their maximum rate from time t' on. Because of the possibly larger processor demand in $[t', t]$, the length of the deadline-t busy period cannot decrease. Thus the τ_i 's instance still misses its absolute deadline at time t.

Finally, let also τ_i be released synchronously from time t'. Once again, the processor demand in $[t', t]$ can only increase. Let t'' be the largest absolute deadline of any task before or at t. The processor demand in $[t', t'']$ coincides with that in $[t', t]$. Hence the deadline-t'' busy period which starts at t', is larger than $t - t' \geq t'' - t'$, that is, we have an absolute deadline missed at time t''. ❑

From Lemma 1, the following properties hold:

- Consider a NSC starting at time *0* (a critical instant), based on a check of all the absolute deadlines. At any time during the schedule, let t' be the beginning of a deadline-t busy period. If

$t' > 0$, any task instance released after t' with an absolute deadline smaller or equal to t cannot miss its absolute deadline if not detected before (otherwise this would contradict Lemma 1). It follows that it is not necessary to check these instances in the interval $[t', t]$.

- the contrary holds, it is sufficient to check the instances of a given task τ_i in the longest synchronous deadline busy period.

As shown in Annex A, for any task we can compute the length L_i of the longest deadline busy period relative to an absolute deadline of a τ_i 's instance. Being the synchronous processor busy period, the largest one, (also shown in Annex A), we necessarily have $L_i \leq L$. Note that:

- L_i depends on the pattern(s) of arrival considered, e.g. for the feasibility analysis that we study here, only the synchronous pattern of arrival is considered (L_i is called L_i^s in such a case).

- the recursive procedure that we propose in Annex A to compute L_i^s can be costly, for specific task sets, in comparison to a simple check of $h(t) \leq t$ on a limited interval (we leave the question of finding a faster procedure to compute L_i^s as an open question to the interested reader). Whatever the case, it is possible to maximise L_i^s by L and, as will be explained in Section 3.1.3, the use of the deadline busy period is more relevant for the analysis of the worst-case response times.

Finally, combining these results, an NSC for general task sets is:

Theorem 6 - *Any general task set with $U \leq 1$ is feasible, using EDF, if and only if:*

$$\forall t \in S,\ h(t) \leq t \tag{2}$$

where

$$S = \left(\bigcup_{i=1}^{n} \left\{ kT_i + D_i, k \in \left[0, \left\lfloor \frac{L_i^s - D_i}{T_i} \right\rfloor \right] \right\} \right) \cap [0, min\{L, B_1, B_2\}]$$

$$B_1 = \frac{\sum_{D_i \leq T_i} (1 - D_i/T_i) C_i}{1 - U}$$

$$B_2 = max\left\{ max_{i=1\ldots n}(D_i),\ \frac{\sum_{i=1}^{n} (1 - D_i/T_i) C_i}{1 - U} \right\}$$

L_i^s is the length, for task τ_i, of its synchronous deadline busy period and L the length of the synchronous processor busy period (see Annex A for the computation).

Proof. Condition (2) is clearly necessary. That it is sufficient is proven by several facts.

The generalization of Theorem 3 and Theorem 4 tells us that if the task set is not feasible, then there is an instant t ($t < L$, $t < B_1$ and $t < B_2$) such as condition (2) is not verified. L comes from [SPU96], B_2 from [ZS94] and B_1 is obtained using an algebraic manipulation similar to that used in Theorem 4. We have

$$t < h(t) \le \sum_{D_i \le t} \left(1 + \frac{t - D_i}{T_i}\right) C_i = t \sum_{D_i \le t} \frac{C_i}{T_i} + \sum_{D_i \le t} \frac{T_i - D_i}{T_i} C_i \le tU + \sum_{D_i \le T_i} \frac{T_i - D_i}{T_i} C_i,$$

from which, if $U < 1$, we can conclude

$$t < \frac{\sum_{D_i \le T_i} (1 - D_i/T_i) C_i}{1 - U}.$$

Furthermore, from Lemma 1 and Annex A, we know that for any task τ_i we only need to check the absolute deadlines of its instances up to s L_i^s, the length of the largest synchronous deadline-d busy period involving a τ_i 's instance with absolute deadline d.

Finally, Zheng and Shin [ZS94] proved that we can ignore the instants not corresponding to any absolute deadline. ❑

Note that condition (2) has a pseudo-polynomial times complexity since L_i^s is upper bounded by L, whose length is pseudo polynomial whenever $U \le c$, with c a positive constant smaller than 1 (see Annex A).

3.1.3 Worst-Case Response times

Contrary to our intuition, and to what happens in fixed priority systems, the worst-case response time of a general task set scheduled by EDF are not necessarily obtained with a synchronous pattern of arrival. In [SPU96], Spuri shows the following lemma.

Lemma 2 – ([SPU96]) *The worst-case response time of a task τ_i is found in a busy period in which all tasks but τ_i are released synchronously and at their maximum rate.*

In practice, in order to find r_i, the worst-case response time of τ_i, we need to examine several scenarios in which for a given a, τ_i has an instance released at time a, while all other tasks are released synchronously ($s_j = 0$, $\forall j \ne i$). In general, τ_i may also have other instances released earlier than a. In particular, its start time is $s_i = a - \lfloor a/T_i \rfloor T_i$.

Given a value of the parameter a, the response time of the τ_i's instance released at time a is $r_i(a) = max\{C_i, L_i(a) - a\}$, where $L_i(a)$ is the length of the busy period, computed by means of

the iterative computation $L_i^{(0)}(a) = 0$, $L_i^{(m+1)}(a) = W_i(a, L_i^{(m)}(a)) + (1 + \lfloor a/T_i \rfloor)C_i$, where

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} min\left\{\left\lceil \frac{t}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right\} C_j.$$

(for an explanation of the formula the interested reader is referred to Annex 1 or [SPU96]).

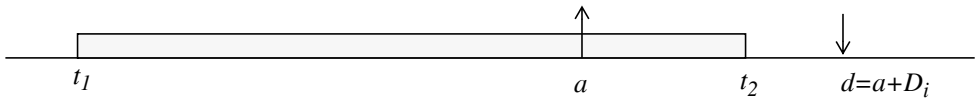
Similarly to what was done in Section 3.1.2 to establish a task set feasibility, the result of Lemma 2 can be improved by using the notion of deadline busy period. In particular, the busy period mentioned in Lemma 2 is always one in which "only instances with absolute deadlines less than or equal to d execute", that is, it is indeed a deadline busy period.

Lemma 3 - The worst-case response time of a task τ_i is found in a deadline busy period for τ_i in which all tasks but τ_i are released synchronously from the beginning of the deadline busy period and at their maximum rate**.**

Proof. Consider a τ_i's instance with release time a and absolute deadline $d = a + D_i$, respectively (see figure 1). Let t_2 be its completion time, according to the EDF scheduling algorithm. Let t_1 be the last time before t_2, such as there are no pending instance with arrival time earlier than t_1 and absolute deadline less than or equal to d (note that $t_2 = L_i(a)$ if $t_1 = 0$).

Since the τ_i's instance is released at time a, $t_1 \leq a$. Furthermore, by choice of t_1 and t_2, t_1 must be the arrival time of a task's instance and there is no idle time in $[t_1, t_2]$. That is $[t_1, t_2]$ is a busy period in which only instances with deadlines less than or equal to d execute, i.e., it is indeed a deadline-d busy period.

Consider now the scenario in which all tasks but τ_i are released synchronously and at their maximum rate from time t_1 on. Because of the larger workload and processor demand in the interval $[t_1, d]$, the completion time of the τ_i's instance considered can only increase. Thus, the response time of the τ_i's instance considered cannot decrease. ❑



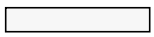

deadline-d busy period

Figure 1

This lets us restrict the interval where the worst-case response time of τ_i has to be looked for, to $[0, L_i]$ with L_i being the maximum length of a deadline busy period, for τ_i (see Annex A for an exact computation of L_i in presence of Lemma 3's patterns of arrival). Note that, contrary to what happens in Section 3.1.2, the computation of L_i might improve significantly the worst-case response time analysis since it already makes use of a recursive expression and since the following property holds (if the tasks are sorted by increasing relative deadline): $\forall i \in [1, n-1]$, $L_i \leq L_{i+1}$ (see Annex A).

According to Lemma 3 and to the subsequent considerations, the significant values of the parameter a are in the interval $[0, L_i]$. Furthermore, also within this interval we can restrict our attention to the points where either $s_i = 0$ or $a + D_i$ coincides with the absolute deadline of another instance, which correspond to local maxima of $L_i(a)$. That is, $a \in A \cap [0, L_i]$, where:

$$A = \bigcup_{j=1}^{n} \{kT_j + D_j - D_j, k \in \aleph\}.$$

Finally, a general task set is feasible, using EDF, if and only if:

$$\forall i \in [1, n], r_i = max_{a \geq 0}\{r_i(a)\} \leq D_i. \quad (3)$$

Note that, similarly to the feasibility condition, the computation of worst-case response time has a pseudo-polynomial time complexity whenever $U \leq c$, with c a positive constant smaller than 1 [SPU96].

3.2 Fixed priority driven schedulers

In this section, we briefly summarize the principles of preemptive fixed priority scheduling. These principles were derived in particular from [LL73], [LW82], [JP86], [LEH90], [AUD91] and [TBW94]. Note that, contrary to what is happening in the dynamic priority case, feasibility and response time computation are closely related in the state of the art.

3.2.1 Optimality

Optimality property is limited in this section to the fixed priority driven class of schedulers. In other words, a scheduler x is said to be optimal in the sense that no other fixed priority assignment can lead to a valid schedule which cannot be obtained by x. This limitation reflects the theoretical dominance of the class of dynamic priority driven schedulers.

In the case of $T_i = D_i, \forall i \in [1, n]$, original work of [LL73] establishes the optimality of the Rate Monotonic (**RM**) priority ordering. The priority assigned to tasks by RM is inversely proportional to their period. Thus the task with the shortest period has the highest priority. For task sets with relative deadlines less or equal to periods, an optimal priority ordering has been shown by [LW82] to be the deadline monotonic (**DM**) ordering. The priority assigned to tasks by DM is inversely proportional to their relative deadline. [LEH90] points out that neither RM nor DM priority ordering policies are optimal for general tasks set (i.e. when relative deadlines are not related to the periods). Finally, [AUD91] solves this problem by giving an optimal priority assignment procedure in $O(n^2)$.

Theorem 7 - *([AUD91]) The Audsley priority assignment procedure is optimal for general task sets.*

The procedure first tries to find out if a task with an assigned priority of level *n* is feasible. Audsley proves that if more than one task is feasible with priority level *n*, one can be chosen arbitrarily among the matching tasks. Then the first feasible task is removed from the task set and priority level is decreased by one. The procedure proceeds with the new priority level until either all remaining tasks have been assigned a priority (the task set is then feasible) or for a certain priority level (no task is feasible, then there is no priority ordering for the given task set).

3.2.2 Feasibility condition and worst-case response times

■ Case $T_i = D_i$, $\forall i \in [1, n]$

Theorem 8 - *([LL73]) For a given synchronous periodic task set (with $T_i = D_i, \forall i \in [1, n]$), the RM schedule is feasible if $U \leq n(2^{1/n} - 1)$.*

[LL73] proved the sufficiency of this processor utilization test for tasks assigned priorities according to the RM showing that it is possible to find a least upper bound on the processor utilization. This result gives us a simple O(n) procedure to check the feasibility.

In addition to this approach based on the processor utilization, another interesting approach focused on deriving the worst-case response time r_i of each task τ_i of a given non-concrete task set. This led to the obvious following NSC that unifies the feasibility of a task set with the worst-case response time:

$$\forall i \in [1, n] \qquad r_i \leq D_i.$$

Let us summarize now the main results on worst-case response times computation in several contexts for preemptive fixed priority driven scheduler.

■ **Case $D_i \leq T_i$, $\forall i \in [1, n]$,**

Theorem 9 - *([JP86]) The worst-case response time r_i of a task τ_i of a non-concret periodic, or sporadic, task set (with $D_i \leq T_i$, $\forall i \in [1, n]$) is found in a scenario in which all tasks are at their maximum rate and released synchronously at a critical instant t=0. r_i is computed by the following recursive equation (where hp(i) denotes the set of tasks of higher priority than task τ_i):*

$$r_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j. \qquad (4)$$

In [LL73] tasks are periodic and relative deadlines are equal to periods. [JP86] analysis is more general because it allows relative deadlines to be smaller than periods. The proof shows that Formula (4) is correlated to the synchronous pattern and cannot be worse in the presence of another pattern. The recursion ends when $r_i^{n+1} = r_i^n = r_i$ and can be solved by successive iterations starting from $r_i^0 = C_i$. Indeed, it is easy to show that r_i^n is non decreasing. Consequently, the series converges or exceeds D_i. In the latter case, task τ_i is not schedulable.

■ **General task sets** ($\forall i \in [1, n]$, T_i and D_i are not related)

Theorem 10 - *([LEH90], [TBW94]): The worst-case response time r_i of a task τ_i of a general task set is found in a scenario in which all tasks are at their maximum rate and released synchronously*

at a critical instant t=0. r_i *is computed by the following recursive equation (where hp(i) denotes the set of tasks of higher priority than task* τ_i*):*

$$r_i = \max_{q = 1 \ldots Q} (w_{i,q} - qT_i), \tag{5}$$

where Q is the minimum value such that $w_{i,Q} \leq (Q+1)T_i$ *and*

$$w_{i,q}^{n+1} = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_{i,q}^{n}}{T_j} \right\rceil C_j. \tag{6}$$

The [JP86] analysis is still not sufficient, as it does not consider periods which are smaller than relative deadlines. This limitation is, however, overcome by [LEH90] using the notion of level-*i* busy period defined as the maximum interval of time during which a processor runs tasks of higher or equal priorities than task τ_i (see Annex A). [LEH90] and [TBW94] show that the worst response time r_i of a given task τ_i occurs during its synchronous level-*i* busy period. In that purpose, they show that it is possible to look successively at several windows, each one starting at a particular arrival of task τ_i. If $w_{i,q}$ denotes the width of the busy period starting at time qT_i before the current instance of task τ_i, the analysis can be performed by the recursive Equation (6) that ends when $w_{i,q}^{n+1} = w_{i,q}^{n} = w_{i,q}$.

Finally, the worst response time of task τ_i is then given by Equation (5) where Q is the minimum value such that $w_{i,Q} \leq (Q+1)T_i$ (meaning that the maximum length of the level-*i* busy period has been examined, see Annex A).

Note that the length of the busy periods that need to be examined is bounded by the lowest common multiple of the tasks periods. It is also bounded by: $(\sum_{j \in hp(i) \cup i} C_j)/(T_i - C_i)$ [LS95]. Also, note that the computation of worst-case response time has a pseudo-polynomial time complexity since any level-i busy period is upper bounded by L, whose length is pseudo-polynomial whenever $U \leq c$, with c a positive constant smaller than 1 (see Annex A).

4. Non-preemptive schedulers

Non-preemptive schedulers have received less attention than the preemptive ones and some results such as optimal schedulers, feasibility conditions or worst-case response times need to be established or to be improved. This chapter gives first a justification for using non-idling scheduling algorithms in non-preemptive context. Then it consider the case of dynamic (resp. fixed) priority driven schedulers, see Section 4.2 (resp. Section 4.3). In particular, we will show that non-idling, non-preemptive scheduling is closely related to preemptive scheduling taking account the extra load resulting to priority inversion.

4.1 Idling/non-idling for non-preemptive scheduling

Non-preemption usually makes the problem of feasibly scheduling a set of tasks more difficult than in preemptive context due to the possible priority inversions and the theoretical dominance of non-

preemptive algorithms that use inserted idle times. However, the general problem of finding a feasible schedule in an idling non-preemptive context is known to be NP-complete [GJ79, annex 5].

In the restrictive hypothesis that all tasks have only one instance and share their release times, we have a very simple solution known as Jackson's Rule [STA95]: if we define the lateness of a task instance as the difference between its completion time and its *due date*, the maximum lateness is minimized when "all instances are put in order of non-decreasing due dates." Unfortunately, the result is not useful for systems in which task are non-concrete and recur. In this case the problem of minimizing the maximum lateness becomes NP-hard [STA95].

When the tasks recur, heuristic techniques can be used [MA84], [MOK83], [ZRS87] to reduce the complexity. However, this reduction is achieved at the cost of obtaining a potentially sub-optimal solution. For example (see next figure), non-preemptive, non-idling EDF is not optimal for idling scheduling.

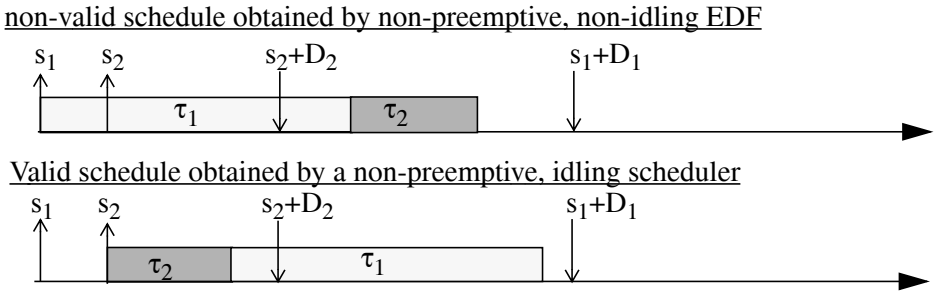


Figure 2: non-preemptive EDF is not optimal in idling context

When the tasks recur and their release times are known a priori, optimal decomposition approaches have been proposed [YUA91], [YUA94] to reduce the complexity by dividing the n tasks into m subsets. Decomposition, however, is not possible for any task sets. In the same context, optimal branch and bound scheduling algorithms [GMR95] have been studied that efficiently limits the cost of an exhaustive search but the theoretical complexity is still in n! in the worst case.

Finally, in presence of non-concret task sets, [HV95] proves formally that the feasibility problem for any periodic task set is NP-Hard in the strong sense when inserted idle times are allowed. It also proves that there cannot exist an optimal on-line inserted idle-time algorithm for scheduling sporadic task sets.

For these reasons, as well as for practical considerations, non-preemptive, idling scheduling algorithms will not be further considered in our context of non-concrete task sets. On the other hand, non-preemptive, non-idling scheduling algorithms will be examined in details in the sequel.

4.2 Dynamic priority driven schedulers

In the following we will first recall the optimality of the EDF algorithm in non-idling, non-preemptive context and second describe useful procedures for the assessment of the feasibility of a task set (see section 4.2.2), as well as for the computation of task worst-case response times (see section 4.2.3).

4.2.1 Optimality

EDF is optimal within the class of non-preemptive non-idling scheduling algorithms, too. The first result concerning the non-preemptive non-idling EDF algorithm was found by Kim and Naghibzadeh [KN80], who showed the optimality of their *relative urgency non-preemptive* strategy, namely non-

preemptive EDF, for the scheduling of a set of sporadic tasks with relative deadlines equal to their periods. A similar result was implicitly given by Jeffay et al. [JSM91], who proposed a sufficient condition for assessing the feasibility of such a set of tasks, which turns out to be also necessary for any given non-idling algorithm. Finally, the optimality of the EDF algorithm was proven in the general model of arbitrary arrival laws and arbitrary relative deadlines by George et al. [GMR95]:

Theorem 11 - *([GRM95]) Non-preemptive non-idling EDF is optimal in the presence of general task set.*

The proof of the theorem is based on a classic interchange argument, by which any valid non-preemptive non-idling schedule is transformed in a polynomial number of steps in a still valid non-preemptive non-idling EDF schedule. In other terms, if there is a valid non-preemptive, non-idling schedule for a given task set, then there is also a valid non-preemptive non-idling EDF schedule. Note that to our knowledge, no optimality result is known within the context of fixed priority scheduling (see Section 4.3.2).

Also note that the Least Laxity First algorithm [MOK83], which at any scheduling decision chooses the task instance with smallest laxity (absolute deadline minus current time minus remaining execution time), and which is optimal in the preemptive context, is no longer optimal when preemption is not allowed. In Figure 3 an example with two tasks whose schedule is valid under EDF, but not under LLF is given. As can be easily remarked, the reason why LLF is no longer optimal is that with respect to this algorithm the priorities of the pending tasks change continuously, thus making necessary the preemption. Being the schedule non-preemptive, an absolute deadline may be missed.

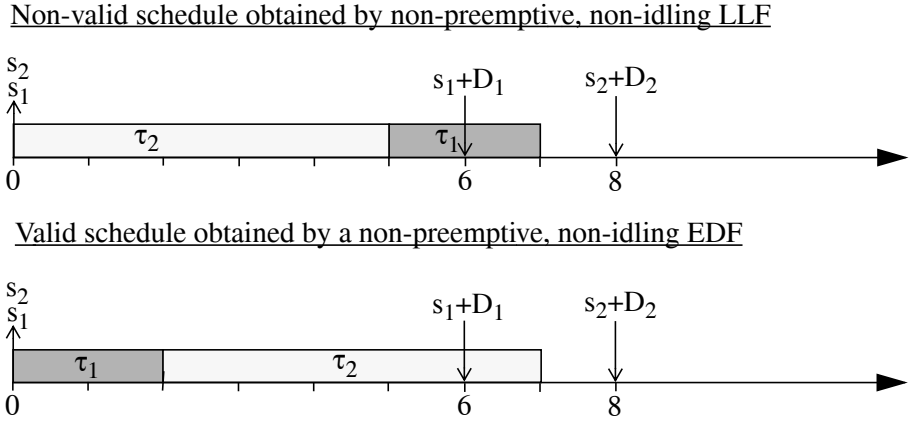


Figure 3: non-preemptive LLF is not optimal

4.2.2 Feasibility

Owing to the optimality of the EDF algorithm, in this section, as well as in the following one, we will assume a non-preemptive non-idling EDF scheduler. In particular, feasibility conditions and worst-case response times, respectively, will be studied for a given task set scheduled according to this model.

■ Case $D_i = T_i$, $\forall i \in [1, n]$

A first feasibility condition for a set of *n* sporadic tasks with relative deadlines equal to their respective periods was given by Kim and Naghibzadeh [KN80]. A similar and well known result, even if in a discrete time model, was then presented by Jeffay et al. [JSM91].

Theorem 12 - *([KN80], [JSM91]) A non-concrete periodic or sporadic task set (with* $D_i = T_i$, $\forall i \in [1, n]$*) is feasible, using EDF, if and only if:*

1) $U \leq 1$,

2) $\forall t \in S, \quad t \geq h(t) + max_{D_i > t}\{C_i - 1\}$,

with the convention that $max_{D_i > t}\{C_i - 1\} = 0$ if $\nexists i: D_i > t$, and

$$S = \left(\bigcup_{i=1}^{n} \{(k+1)T_i, k \in \aleph\} \right) \cap [0, max\{D_i\}],$$

This pseudo-polynomial test is basically similar to the preemptive case. In preemptive context, for any absolute deadline *t*, a NSC is to check that $\forall t, \quad t \geq h(t)$. Nevertheless, in non preemptive context, we must account for an extra load resulting to a priority inversion w.r.t priority *t*. Indeed, even if EDF scheduling policy is considered, due to non-preemption, it might occur that at time *0*, task's instances are pending with their absolute deadline smaller or equal to *t* and at the same time *0* a task instance with an absolute deadline greater than *t* is scheduled. The worst duration of a priority inversion w.r.t priority *t* is then given by $max_{D_i > t}\{C_i - 1\}$, following that for any time $t > max\{D_i\}$, preemptive EDF or non-preemptive EDF scheduling are basically similar (since in that case, $max_{D_i > t}\{C_i - 1\} = 0$). Condition 2 includes this worst case duration of a priority inversion. Given this similarity and as $D_i = T_i$, $\forall i \in [1, n]$ the [LL73] condition ($U \leq 1$ that is condition 1 in this theorem) is sufficient after $max\{D_i\}$. Finally, as for the preemptive case [KN80] proposed to evaluate the processor demand only on the set S of points corresponding to absolute deadlines of task requests (i.e. the set of points where the value of $h(t)$ changes).

■ **General case** ($\forall i \in [1, n]$, T_i and D_i are not related)

Recently, Zheng and Shin [ZS94] focused on message communication systems in which the relative deadlines and the periods are not necessarily related. In that model, they established the following sufficient (but not necessary in our task context) feasibility condition.

Theorem 13 - *([ZS94]) A general message set (with* $U \leq 1$ *) is feasible, using EDF, if:*

$\forall t \in S, \quad t \geq h(t) + C_p - 1,$ with C_p the maximum transmission time of any message and

$$S = \left(\bigcup_{i=1}^{n} \{D_i + kT_i, k \in \aleph\} \right) \cap \left[0, max\left(max_{i=1\ldots n}(D_i), \frac{C_p - 1 + \sum_{l=1}^{n}(1 - D_i/T_i)C_l}{1 - U} \right) \right].$$

As for the preemptive model, they show that the number of checks to do in order to assess the feasibility can be first limited in a bounded interval (using successive overvaluation of $h(t)$) and second to a restricted set of suitable time instants (i.e. the set of points in S where the value of $h(t)$ changes). However, as their model is conceived for message communication systems, the cost of the non-preemptive effect is larger than necessary in our context. Indeed in this theorem, the cost of possible priority inversions, caused by message non-preemptability, is always initiated by C_p (the worst transmission time) and moreover is effective during all the studied interval. In the following we

precisely state a NSC for the feasibility of a general task set. Furthermore, as for the preemptive case, we will see that the notion of *deadline busy period* is still useful.

As remarked in Annex A, as far as the schedule is non-idling, the length L of the *synchronous processor busy period* does not depend on the scheduling algorithm, and is the biggest among all busy periods. This property limits the number of checks to do in order to assess the feasibility of a given task set. Moreover, as for the preemptive context, we can make use of the more specialized notion of deadline busy period to refine our analysis by further restricting the interval of interest. In non-preemptive context, a deadline-d busy period is still a busy period in which only instances with absolute deadlines before or at d are executed (see Annex A for a formal definition). However, since we must take into account the effect of non preemption, it might occur that a priority inversion precedes the deadline-d busy period. It turns out that only the synchronous deadline-d busy periods preceding by the worst possible priority inversion are the busy periods interesting in order to check the feasibility of a task set.

Lemma 4 - Given a general task set, if there is an overflow at time t_2 for a certain pattern of arrival, then there is an overflow in a deadline-t_2 busy period starting at time t_1 such as one task τ_j starts its execution at time $t_1 - B - 1$, where $B = max_{D_j > t_2 - t_1 + C_j - 1}\{C_j - 1\}$ ($B = 0$ when $\forall j, \quad D_j \le t_2 - t_1 + C_j - 1$) and all the other tasks are released at time $t_1 - B$ (see figure 4).

Proof. Assume there is a pattern which causes an instance of task τ_i to miss its absolute deadline at time t_2. Let t'_1 be the last time before t_2, such that there are no pending instances with arrival times earlier than t'_1 and absolute deadline before or at t_2. By choice, t'_1 must be the arrival time of a task instance, and there is no idle time between t'_1 and t_2. Furthermore:

- it might occur that a priority inversion takes place at t'_1, due to the effect of non preemption. Let us call b the duration of such a priority inversion.
- only instances with absolute deadline before or at t_2 are executed in $[t'_1 + b, t_2]$. That is, $t'_1 + b$ is the beginning of a deadline-t_2 busy period, whose length is greater than $t_2 - t'_1 - b$, by the hypothesis of overflow at time t_2.

Let τ_j be the task verifying $B = max_{D_j > t_2 - t'_1}\{C_j - 1\}$ ($B = 0$ when $\forall j \qquad D_j \le t_2 - t'_1$), then B is the length of the worst possible priority inversion if τ_j starts its execution at time $t'_1 - 1$. Indeed, any other priority inversion $b \le max_{D_j > t_2 - t'_1}\{C_j - 1\}$.

Consider now the scenario in which τ_j is released at time $t'_1 - 1$ and all the other tasks but τ_i are released synchronously and at their maximum rate from time t'_1 on. Because of the worst possible priority inversion, the beginning of the deadline-t_2 busy period (that is now in $t_1 = t'_1 + B$) is delayed. Moreover, because of the possibly larger processor demand in $[t'_1, t_2]$, the length of the deadline-t_2 busy period cannot decrease. Thus the τ_i's instance still misses its absolute deadline at time t_2.

Finally, let also τ_i be released synchronously from time t'_1. Once again, the processor demand in $[t'_1, t_2]$ can only increase. Let t'_2 be the largest absolute deadline of any task before or at t_2. The processor demand in $[t'_1, t'_2]$ coincides with that in $[t'_1, t_2]$. Hence the deadline-t'_2 busy period which starts at t_1, is larger than $t_2 - t'_1 \geq t'_2 - t'_1$, that is, we have an absolute deadline missed at time t'_2. ❑

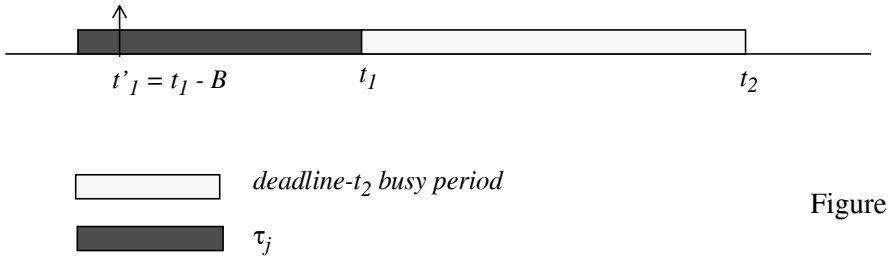


Figure 4

As for the preemptive case, for any task τ_i we can compute, in non-preemptive context and in presence of the pattern of arrival described for Lemma 4, the length L_i^s of the longest synchronous deadline busy period relative to an absolute deadline of a τ_i 's instance (see Annex A).

As will be seen now in the following theorem, the assessment is achieved by checking the absolute deadlines of several task instances within particular busy periods. The length *L* of the synchronous busy period is thus a useful upper bound that lets us limit the number of absolute deadlines to be checked and that can be combined with the bound established by [ZS94] in Theorem 13. Note that the feasibility condition we are going to show is a necessary and sufficient one since it computes exactly the worst cost of possible priority inversions, caused by task non-preemptability. Finally, combining all these results, an NSC for general task sets in non-preemptive context is:

Theorem 14 - *Any general task set with* $U \leq 1$ *is feasible, using EDF, if and only if:*

$$\forall t \in S, \quad t \geq h(t) + max_{D_i > t}\{C_i - 1\}, \tag{7}$$

with the convention that $max_{D_i > t}\{C_i - 1\} = 0$ *if* $\nexists i$*:* $D_i > t$ *where,*

$$S = \left(\bigcup_{i=1}^{n} \left\{ kT_i + D_i, k \in \left[0, \left\lfloor \frac{L_i^s - D_i}{T_i} \right\rfloor \right] \right\} \right) \cap [0, min\{L, B_1, B_2\})$$

$$B_1 = \frac{\sum_{D_i \leq T_i} (1 - D_i/T_i) C_i + max_i\{C_i - 1\}}{1 - U},$$

$$B_2 = max\left(max_{i=1\ldots n}(D_i), \frac{\sum_{l=1}^{n} (1 - D_i/T_i) C_i}{1 - U} \right),$$

L_i^s is the length, for task τ_i, of its synchronous deadline busy period and L the length of the synchronous processor busy period.

Proof. Condition (7) is clearly necessary. Its sufficiency comes from several facts.

If the task set is not feasible, than there exists an instant t ($t < L$, $t < B_1$ and $t < B_2$) such that (7) is not verified. B_2 comes from Theorem 13 and B_1 can be obtained using similar algebraic manipulations. In fact, if the condition is not satisfied for some value *t*, then

$$t < \sum_{D_i \le t} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i + max_{D_i > t}\{C_i - 1\} \le \sum_{D_i \le t} \left(1 + \frac{t - D_i}{T_i} \right) C_i + max_{D_i > t}\{C_i - 1\}$$

Then:

$$t < t \sum_{D_i \le t} \frac{C_i}{T_i} + \sum_{D_i \le d} \frac{T_i + D_i}{T_i} C_i + max_{D_i > t}\{C_i - 1\} \le tU + \sum_{D_i \le T_i} \frac{T_i + D_i}{T_i} C_i + max_i\{C_i - 1\}$$

from which we finally have: $t < \dfrac{\sum_{D_i \le T_i} (1 - D_i / T_i) C_i + max_i\{C_i - 1\}}{1 - U}$.

Furthermore, from Lemma 4 and Annex A, we know that for any task τ_i we only need to check the absolute deadlines of its instances up to time L_i^s, the length of the largest synchronous deadline-d busy period involving a τ_i's instance with absolute deadline d. Being the synchronous processor busy period the largest one (also shown in Annex A), we necessarily have $L_i^s \le L$, and then L is also an upper bound.

Finally, Zheng and Shin [ZS94] proved that we can ignore the instants not corresponding to any absolute deadline. ❑

Ccondition (7) has a pseudo-polynomial time complexity since L_i^s is upper bounded by L, whose length is pseudo polynomial whenever $U \le c$, with c a positive constant smaller than 1 (see Annex A).

Remark: Note that owing to the general optimality of the preemptive EDF algorithm with respect to uniprocessor scheduling, the feasibility of a task set under non-preemptive EDF scheduling implies the feasibility of the same task set under preemptive EDF scheduling. The opposite is not true. Note also that the feasibility of a task set under non-preemptive fixed priority scheduling does not imply its feasibility under preemptive fixed priority scheduling.

4.2.3 Worst-case response times

In the previous section we have seen that the feasibility check of a task set scheduled by a non-preemptive non-idling EDF scheduler is very similar to the preemptive case. As expected, the similarity is valid also for the computation of task worst-case response times. There are, however, two differences which are worth to mentioning:

- The first one has been already described in Theorem 14. Owing to the absence of preemption, a task instance with later absolute deadline can possibly cause a priority inversion, which must be accounted for.

- The second difference is in some way subtler. Always owing to the non-preemptability of any task instance execution, our attention will be on the busy period preceding the execution start time of the instance, and not on the busy period preceding its completion time, as is the case in the preemptive model. As will be clearer along the proof of the following lemma, this slightly changes the way worst-case response times are computed.

Before describing the details of the worst-case response times computation, we first need to characterize the scenarios, and in particular the consequent deadline busy periods, which provide the local response times maxima.

Lemma 5 - The worst-case response time of a task τ_i is found in a deadline busy period for τ_i in which τ_i has an instance released at time a (and possibly others released before), all tasks with relative deadline smaller than or equal to $a + D_i$ are released from time $t = 0$ on at their maximum rate, and finally a further task with relative deadline greater than $a + D_i$, if any, has an instance released at time $t = -1$.

Proof. Consider a scenario in which τ_i has an instance with arrival time t_1 and absolute deadline $t_2 = t_1 + D_i$ (see figure 5). Let t_4 be the instance execution start time, according to the non-preemptive non-idling EDF schedule. Finally, let t_3 be the last time before or at t_1 such that there are no pending instances with arrival times before t_3 and absolute deadline before or at t_2.

By choice of t_3 and t_4, t_3 must be the release time of a task's instance, and there cannot be idle time between t_3 and t_4. That is, the execution of the τ_i's instance arrived at time t_1 is preceded by a busy period of those instances released between t_3 and t_4, and that have absolute deadlines before or at t_2 (the mentioned busy period is then a deadline-t_2 busy period), plus, owing to the non-preemptability of executions, at most one other instance released before t_3 and having absolute deadline after t_2.

Consider now the scenario in which:

- all tasks but τ_i with relative deadline less than or equal to $t_2 - t_3$ are released from time $t = 0$ on at their maximum rate,

- τ_i is released at time $a - \left\lfloor \frac{a}{T_i} \right\rfloor T_i, \quad a - \left(\left\lfloor \frac{a}{T_i} \right\rfloor - 1 \right) T_i, \quad \ldots, \quad a$ (with $a = t_1 - t_3$,),

- and the task τ_j, if any, which attains the maximum value of $max_{D_j > t_2 - t_3}\{C_j - 1\}$ is released at time $t = -1$. That is, τ_j cause the worst possible priority inversion w.r.t. the absolute deadline $a + D_i$.

In the new scenario, the workload in the interval preceding the start time of the τ_i's instance released at time $a = t_1 - t_3$ cannot be less than that between t_3 and t_4 in the previous scenario. Hence the busy period preceding this τ_i's instance execution cannot be shorter since it includes the worst-case priority inversion w.r.t. the absolute deadline $a + D_i$ and the largest deadline-$a + D_i$ busy period preceding it. That is, its relative response time cannot diminish. ❑

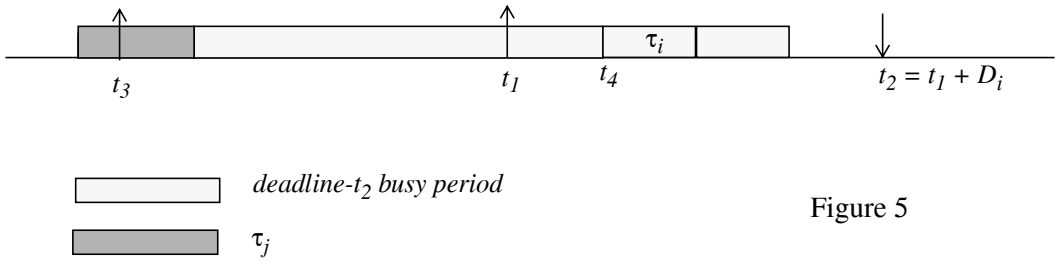


Figure 5

As suggested by the lemma, if the length of the busy period starting at time $t = 0$ and preceding the execution of the τ_i's instance released at time a is termed $L_i(a)$, the response time of the instance is

$$L_i(a) + C_i - a.$$

Since the computation of $L_i(a)$ may occasionally give a value smaller than a, more generally we have

$$r_i(a) = max\{C_i,\ L_i(a) + C_i - a\}.$$

The length $L_i(a)$ can be determined by finding the smallest solution of the equation

$$t = max_{D_j > a + D_i}\{C_j - 1\} + \overline{W}_i(a,t) + \left\lfloor \frac{a}{T_i} \right\rfloor C_i, \qquad (8)$$

where the first term on the right side accounts for the worst-case priority inversion w.r.t. the absolute $a + D_i$, while the second and the third terms represent the time needed to execute the largest deadline-$a + D_i$ busy period that precede the execution of the τ_i's instance released at time a. More precisely, the second term is the time needed to execute some instances of tasks other than τ_i with absolute deadlines smaller than or equal to $a + D_i$ and release times before this τ_i's instance execution start time. Finally, the third term is the time needed to execute the τ_i's instances released before a.

The rationale of the equation is to compute the time needed by the τ_i's instance released at time a to get the processor: every other higher priority instance released before this event will be executed earlier, thus its execution time must be accounted for. For the same reason, the function $\overline{W}_i(a,t)$ must account for all higher priority instances released in the interval $[0,t]$, thus also including those

possibly released at time t. For any task τ_j, the maximum number of instances released in $[0,t]$ is $1 + \lfloor t/T_j \rfloor$.

However, at most $1 + \lfloor (a + D_i - D_j)/T_j \rfloor$ among them can have an absolute deadline before or at $a + D_i$. It follows that

$$\overline{W}_i(a, t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} min\left\{1 + \left\lfloor \frac{t}{T_j} \right\rfloor, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right\} C_j.$$

Being $\overline{W}_i(a,t)$ a monotonic non-decreasing step function, the smallest solution of Equation (8) can be found by using the usual fixed point computation:

$$\begin{cases} L_i^{(0)}(a) = 0 \\ L_i^{(m+1)}(a) = max_{D_j > a + D_i}\{C_j - 1\} + \overline{W}_i(a, L_i^{(m)}(a)) + \left\lfloor \frac{a}{T_i} \right\rfloor C_i. \end{cases}$$

According to the argument of Lemma 5, we have defined the response time relative to a, $r_i(a)$, as a function of the busy period length $L_i(a)$ which is upper bounded by L, the length of the synchronous busy period (see Annex A). Hence we conclude that the computation of $r_i(a)$ can be coherently limited to values of a smaller than L. That is, the worst-case response time of τ_i is finally

$$r_i = max\{r_i(a): 0 \leq a < L\}.$$

The number of evaluations of $r_i(a)$ necessary to compute r_i can be further reduced by observing that the right side of Equation (8) is a step function whose discontinuities in a are for values equal to $kT_j + D_j - D_i$, for some task τ_j and some integer *k*. The significant values of a in the interval $[0,L)$ can be reduced accordingly.

Moreover, as for the preemptive case, it is possible to restrict the interval where the worst-case response time of τ_i has to be looked for, to $[0, L_i]$ with L_i being the maximum length of a deadline busy period, for τ_i in non-preemptive context (see Annex A for an exact computation of L_i in presence of Lemma 5's patterns of arrival). Once again, note that, contrary to what happens in the feasibility section, the computation of L_i might improve significantly the worst-case response times analysis since it already makes use of recursive expression and since the following property holds (if the tasks are sorted by increasing relative deadline): $\forall i \in [1, n-1]$, $L_i \leq L_{i+1}$ (see Annex A).

Note that, similarly to the feasibility condition, the computation of worst-case response times has pseudo-polynomial time complexity since L_i is upper bounded by L, whose length is pseudo polynomial whenever $U \leq c$, with c a positive constant smaller than 1 (see Annex A).

4.3 Fixed priority driven schedulers

This chapter first concentrates on feasibility conditions and worst-case response times computation for general task sets. Some existing results in continuous scheduling are adapted to the context of discrete scheduling. Optimality in fixed priority driven schedulers is then studied. We will notably show that the optimal priority assignment algorithm proposed by [AUD91] in preemptive scheduling is still valid when extended to the non-preemptive context.

4.3.1 Feasibility Condition and Worst-Case Response times

Contrary to the preemptive context, less results are known about fixed priorities non-preemptive scheduling. In the context of continuous scheduling, where tasks parameters and time are allowed to be non integer, one can derive from [THW94] and [TBW95] the following condition for the feasibility of a task set with arbitrary priorities:

Let $\tau = \{\tau_1, \ldots, \tau_n\}$ be a general task set with arbitrary priorities assigned by some algorithm. As in the preemptive context, the feasibility condition is based on the computation of the worst-case response time of each task τ_i, and by comparing its value with the relative deadline D_i. The non-preemptability of the schedule is taken into account by considering a blocking factor $B_i = max_{j \in lp(i)}\{C_j\}$, where $lp(i)$ is the subset of indexes that identify the tasks with lower priority than τ_i. The worst-case response time r_i of τ_i can thus be figured out by means of the following recursive equation:

$$r_i = \max_{q = 0 \ldots Q} \{w_{i,q} + C_i - qT_i\} \qquad \text{where} \qquad w_{i,q} = qC_i + \sum_{j \in hp(i)} \left\lceil \frac{w_{i,q} + \Upsilon_{res}}{T_j} \right\rceil C_j + B_i$$

Q is the smallest value such that $w_{i,Q} + C_i \leq (Q+1)T_i$, Υ_{res} is the resolution with which time is measured, and $hp(i)$ is the subset of indexes that identify the tasks with higher priority than τ_i.

Although continuous scheduling is more general than discrete scheduling, [BHR90] argue in favor of discrete scheduling, showing first that it is reasonable to restrict task parameters to be integer, as any scheduler is limited to scheduling in multiples of some discrete time unit, and as task parameters are expressed in that time unit. Second, they show that once the input has been restricted to be integer, a valid continuous schedule exists if and only if a valid discrete schedule exists, i.e we can consider without loss of generality that tasks are scheduled at integer times.

If we limit our attention to discrete contexts, the previous feasibility condition is still sufficient but no longer necessary. For example, let the task set

$$\tau = \{\tau_1(C_1 = 2, T_1 = 5, D_1 = 3), \tau_2(C_2 = 2, T_2 = 10, D_2 = 10)\}$$

be scheduled according to DM. The worst-case response times computed by using the previous formula with $\Upsilon_{res} = 1$ in our context are $r_1 = 4$ and $r_2 = 4$. The task set is declared unfeasible, although it is easy to verify that it is indeed feasible. Namely, according to Theorem 15, which is later shown, $r_1 = 3$ and $r_2 = 4$.

Establishing a necessary and sufficient feasibility condition for any general task set with arbitrary fixed priorities, essentially means to develop a procedure for exactly computing the task worst-case response

times. In order to achieve this goal, we first show that the notion of level-i busy period, introduced by [LEH90], is also useful in the non-preemptive context. As for the dynamic case, the main differences with the preemptive context are:

- Owing to the absence of preemption, a task instance with later absolute deadline can possibly cause a priority inversion, which must be accounted for.
- Always owing to the non-preemptability of any task instance execution, our attention will be on the busy period preceding the execution start time of the instance, and not on the busy period preceding its completion time, as is the case in the preemptive model.

Let $\tau = \{\tau_1, \ldots, \tau_n\}$ be a general task set with arbitrary fixed priorities.

Lemma 6 – *The worst-case response time of τ_i is found in a level-i busy period obtained by releasing all tasks τ_j with $j \in hp(i) \cup \{i\}$ synchronously from time $t = 0$, and by releasing the longest task τ_k with $k \in lp(i)$, if any, at time $t = -1$.*

Proof. Consider the schedule produced by the non-preemptive highest priority first algorithm for a given scenario (see figure 6). Let t_2 be the completion time of one of the τ_i's instances. Let t_1 be the last time before t_2 such that there are no pending instances before t_1 with priority higher or equal to that of τ_i. By definition, there is no idle time in $[t_1, t_2]$, and the only tasks that have instances executed in $[t_1, t_2]$ are those with indexes in $hp(i) \cup \{i\}$. In addition, the instance of a lower priority task, if any, may execute at t_1 owing to the non-preemptability of executions (note that in this case the lower priority instance must have been released before t_1). The interval between the completion time of this instance (t_1, if there is no such instance) and t_2 is a level-i busy period.

The response time of the τ_i's instance considered can be possibly worsened in the following ways. If all instances of τ_i in $[t_1, t_2]$ are actually released from t_1 at their maximum rate, each execution finishes at the same time, but has possibly a larger response time. Similarly, if all tasks with higher priority than τ_i are released synchronously from t_1, the number of higher priority instances cannot decrease, thus giving a possibly longer response time for τ_i's instances. Finally, if τ_k is the task with the maximum execution time among all tasks with lower priority than τ_i ($k \in lp(i)$), by releasing an instance of τ_k at time $t_1 - 1$, the effect of non-preemption is maximized, thus possibly worsening the response time of τ_i's instances. By substituting t_1 with 0 and t_2 with $t_2 - t_1$ we have the thesis.❑

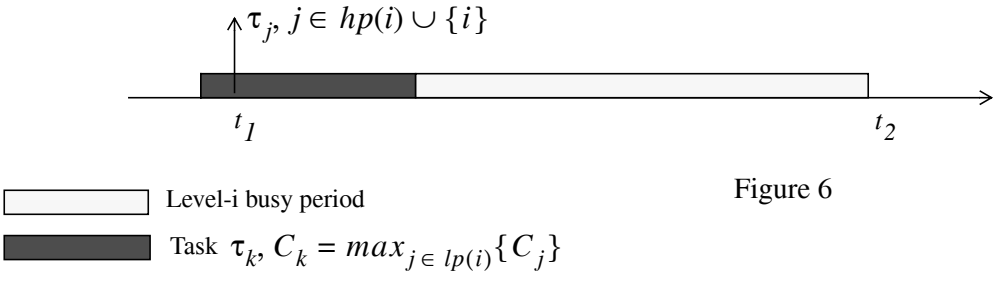


Figure 6

The result of the previous lemma lets us extend the approach developed by [AUD91] and [LEH90], to figure out the worst-case response times of a task τ_i in the non-preemptive context. Unless stated otherwise, we will only examine scenarios as described in Lemma 6.

Theorem 15 - *Given a general task set $\tau = \{\tau_1, \ldots, \tau_n\}$ with arbitrary fixed priorities, the worst-case response time of any task τ_i is given by*

$$r_i = max_{q = 0, \ldots, Q}\{w_{i,q} + C_i - qT_i\},$$

where

$$w_{i,q} = qC_i + \sum_{j \in hp(i)} \left(1 + \left\lfloor \frac{w_{i,q}}{T_j} \right\rfloor\right) C_j + max_{k \in lp(i)}\{C_k - 1\}, \quad (9)$$

and $Q = \lfloor L_i / T_i \rfloor$, where L_i is the length of the longest level-i busy period in non-preemptive context (see Annex A.2).

Proof. Given a task τ_i, consider its instance released at time qT_i. $w_{i,q}$ is the smallest time such that the workload in the interval $[0, w_{i,q}]$ due to all task instances which precede the execution of the τ_i 's instance considered is maximum and equal to $w_{i,q}$, i.e no other task instance can delay the $(q+1)$-th instance of τ_i at time $w_{i,q}$. In Equation (9), qC_i stands for the duration of the q instances of τ_i released before qT_i. $\sum_{j \in hp(i)}(1 + \lfloor w_{i,q}/T_j \rfloor)C_j$ stands for the maximum workload of tasks with higher priority than τ_i in the interval $[0, w_{i,q}]$ and $max_{k \in lp(i)}\{C_k - 1\}$ is the maximum delay resulting from tasks with lower priority than τ_i (worst-case according to Lemma 6). Once it has gained the processor at time $w_{i,q}$, the $(q+1)$-th instance of τ_i completes its execution by time $w_{i,q} + C_i$. Its response time is therefore $w_{i,q} + C_i - qT_i$.

L_i is the maximum length of any level-i busy period (see Section A.2). Thus, according to Lemma 6, we do not need to examine instances released after L_i, that is, the last one to be considered is that released at time $\lfloor L_i/T_i \rfloor T_i$. The worst-case response time of τ_i is finally

$$r_i = max_{q = 0, \ldots, Q}\{w_{i,q} + C_i - qT_i\}. \quad \square$$

Note that, similarly to preemptive case, the computation of worst-case response times has a pseudo-polynomial time complexity, since L_i is upper bounded by L, whose length is pseudo polynomial whenever $U \leq c$, with c a positive constant smaller than 1 (see Annex A).

4.3.2 Optimality

■ **Case** $D_i \leq T_i$, $\forall i \in [1, n]$

Contrary to the preemptive case, when preemption is not allowed DM is no longer an optimal priority assignment in this case. However, we first show that the optimality remains if slightly stronger conditions are imposed.

To prove that DM is no longer optimal, we only need to give a counter example. Let $\tau = \{\tau_1, \tau_2, \tau_3\}$ be a periodic task set with $\tau_1(C_1=3, T_1=5, D_1=5)$, $\tau_2(C_2=2, T_2=10, D_2=6)$, and $\tau_3(C_3=1, T_3=10, D_3=7)$. Assume the priorities are assigned according to DM (i.e., τ_1, τ_2 and τ_3 have decreasing priorities).

In a synchronous scenario, the first occurrence of task τ_3 is not executed by time $D_3 = 7$ (see Figure 7). Its response time (see Theorem 15) is $r_3 = 2C_1 + C_2 + C_3 > D_3$, hence the deadline is missed. The schedule is not valid. Yet, it is easy to see that by assigning priorities in decreasing order to τ_1, τ_3, and τ_2, respectively, the schedule becomes valid, and the task set feasible. Indeed, we have:

$r_1 = C_1 + (C_2 - 1) \leq D_1$,

$r_3 = C_1 + C_3 + (C_2 - 1) \leq D_3$,

$r_2 = C_1 + C_2 + C_3 \leq D_2$.

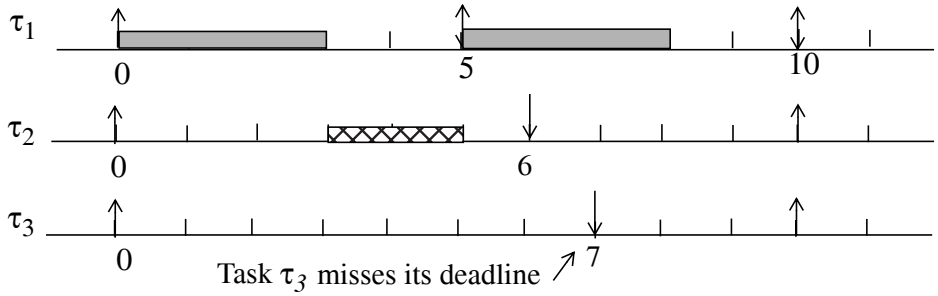


Figure 7: τ is not feasible with DM's priority assignment

Remark: the schedule produced by the preemptive DM with the same synchronous pattern would have led similarly to a deadline miss for task τ_3. Since DM is optimal in preemptive context when $\forall i = 1 \ldots n$, $D_i \leq T_i$, we may conclude that τ is not feasible when preemption is allowed. Thus we have found a task set that is feasible in non-preemptive context with a fixed priority assignment, but not feasible in preemptive context.

Let us show now that the optimality of DM is kept when deadline and execution time orders are similar.

Theorem 16 - *DM is an optimal priority assignement for periodic or sporadic task sets with* $D_i \le T_i, \forall i$, *if* $D_i < D_j \Rightarrow C_i \le C_j$.

Proof. We will show that whenever a valid schedule exists, a valid DM schedule exists, too. In that purpose, let $\tau = \{\tau_1, \ldots, \tau_n\}$ be a set of n tasks with a given feasible priority assignment. Let τ_i and τ_j be two tasks of adjacent priorities, with τ_j the highest priority one (i.e., $hp(i) = hp(j) \cup \{j\}$). Assume that $D_i < D_j$. By hypothesis we also have $C_i \le C_j$.

As the task set is schedulable, the worst case response time of each task is less than or equal to its relative deadline. Furthermore, as $\forall i, D_i \le T_i$, the worst-case response time of any task is found in its first instance ($Q = 0$ in Theorem 15). Thus we have:

$r_j = w_j + C_j$, with $w_j = \sum_{k \in hp(j)} (1 + \lfloor w_j / T_k \rfloor) C_k + max_{k \in lp(j)} \{C_k - 1\}$, and

$r_i = w_i + C_i$, with $w_i = \sum_{k \in hp(i)} (1 + \lfloor w_i / T_k \rfloor) C_k + max_{k \in lp(i)} \{C_k - 1\}$.

Note that $w_i < w_i + C_i \le D_i < D_j \le T_j$, so $1 + \lfloor w_i / T_j \rfloor = 1$.

Let us swap the priorities of τ_i and τ_j, so as to have a deadline monotonic ordering between the two tasks. Since the priority of τ_i has raised, its new worst-case response time cannot increase, that is, the schedule remains valid for τ_i. Vice versa, the priority of τ_j has lowered, so its worst-case response time may increase. The new value is $r_j' = w_j' + C_j$, with w_j' equal to the smallest solution of the equation $w = \sum_{k \in hp'(j)} (1 + \lfloor w / T_k \rfloor) C_k + max_{k \in lp'(j)} \{C_k - 1\}$, with $hp'(j) = hp(j) \cup \{i\}$. If we evaluate the right term of the equation in $w = w_i - C_j + C_i$, we have

$$\sum_{k \in hp'(j)} \left(1 + \left\lfloor \frac{w_i - C_j + C_i}{T_k} \right\rfloor\right) C_k + max_{k \in lp'(j)} \{C_k - 1\} \le$$
$$\sum_{k \in hp(j) \cup \{i\}} \left(1 + \left\lfloor \frac{w_i}{T_k} \right\rfloor\right) C_k + max_{k \in lp(j) - \{i\}} \{C_k - 1\} =$$
$$\sum_{k \in hp(j)} \left(1 + \left\lfloor \frac{w_i}{T_k} \right\rfloor\right) C_k + C_i + max_{k \in lp(i)} \{C_k - 1\} =$$
$$\sum_{k \in hp(j)} \left(1 + \left\lfloor \frac{w_i}{T_k} \right\rfloor\right) C_k + C_j + max_{k \in lp(i)} \{C_k - 1\} + C_i - C_j =$$
$$\sum_{k \in hp(i)} \left(1 + \left\lfloor \frac{w_i}{T_k} \right\rfloor\right) C_k + max_{k \in lp(i)} \{C_k - 1\} + C_i - C_j = w_i + C_i - C_j.$$

It follows that $w_j' \le w_i + C_i - C_j$, hence $r_j' = w_j' + C_j \le w_i + C_i \le D_i < D_j$, that is, also τ_j remains feasible with the new priority assignment. Since the worst-case response times of all other tasks are not affected by the "priority swap", the new priority assignment is globally feasible. A deadline monotonic feasible priority assignment can be finally achieved with a finite number of similar steps. ❑

■ **General task sets** ($\forall i \in [1, n]$, T_i and D_i are not related)

An optimal priority assignment for general task sets scheduled in preemptive systems has been described in [AUD91]. We want to prove that the strategy is optimal also in non-preemptive systems. In order to do this, we first need to show that decreasing the priority of a task is harmless for the other tasks.

Lemma 7 - *Decreasing the priority of a task can only decrease or leave unchanged the response times of the other tasks.*

Proof. Let $\tau = \{\tau_1, \ldots, \tau_n\}$ be a set of n general tasks with increasing priorities. Let τ_k be the task whose priority is to be decreased, and let $\tau' = (\tau_1, \ldots, \tau_{k-1}, \tau_{k+1}, \ldots, \tau_m, \tau_k, \tau_{m+1}, \ldots, \tau_n)$ be the new priority ordering. In τ' we can distinguish three subsets: $A = \{\tau_1, \ldots, \tau_{k-1}\}$, $B = \{\tau_{k+1}, \ldots, \tau_m\}$, and $C = \{\tau_{m+1}, \ldots, \tau_n\}$.

The worst-case response times of the tasks in A and C are not affected by the new priority ordering. Indeed, their priorities do not change, while the new priority of task τ_k produces the same worst-case scenario (see Theorem 15). Thus the response times of those tasks are unchanged.

The tasks of subset B see their priorities to decrease by 1 (having more priority) thus possibly decreasing their worst-case response times (obvious from the formula of Theorem 15). ❑

The following theorem is inspired from [AUD91], where it is proven in a preemptive context. Let τ be a task set of n general tasks, $\tau = \{\tau_1, \ldots, \tau_n\}$, and let $\Phi^\tau(\,)$ be a particular priority assignment function, such that for any task τ_i, $i = 1 \ldots n$, $\Phi^\tau(\tau_i)$ is the priority of the task τ_i in τ.

Theorem 17 - *Let $sb = \{\tau_i, \ldots, \tau_n\}$ be a subset of τ. Suppose that tasks in sb are feasible when $\forall k = i \ldots n$, task τ_k is assigned priority k. If there exists a priority assignment function $\Phi^\tau(\,)$ that enables τ to be feasible then there exists a priority assignment function that assigns the tasks in sb priority $i, i+1, \ldots, n$ such that τ is still feasible.*

Proof. We prove the theorem by induction. Suppose τ_n be feasible at priority n. Suppose that a feasible priority ordering $\Phi^\tau(\,)$ exists that assigns τ_n priority $m < n$. By Lemma 7, if the priority of τ_n is changed from m to n the response times of the other tasks is not worsened. As τ_n is feasible at priority level n, the resulting priority ordering is still feasible.

Suppose the property true for tasks $\tau_{i+1}, \ldots, \tau_n$: those tasks reassigned priority $i+1, \ldots, n$ still enable τ to be feasible. Let $\Phi^\tau(\,)$ to be updated according to the new priority assignment.

Suppose now that τ_i is feasible at priority i but has a priority $\Phi^{\tau}(\tau_i) < i$. By Lemma 7, changing the priority $\Phi^{\tau}(\tau_i)$ of τ_i to i cannot increase the response times of the other tasks, and by hypothesis τ_i is feasible at priority i. Hence τ is still feasible when tasks in sb are assigned priorities $i, i+1, \ldots, n$. ❑

From the above theorem, we can derive an optimal priority assignment based on the approach described in [AUD91], which is thus also valid in non-preemptive context.

Let $\tau = \{\tau_1, \ldots, \tau_n\}$ be a general task set. In order to determine an optimal priority assignment for those tasks, we proceed in the following way. We check if at least one task is feasible (according to Theorem 15) if assigned priority n. Two cases are possible:

- no task is feasible, then clearly no feasible priority assignment exists for τ.
- at least one task is feasible. If several tasks are feasible, we can choose one at random. Indeed, by Theorem 17, if a feasible priority assignment exists then one will exist with priority n for the selected task.

We then repeat the steps to priority $n-1, \ldots, 1$, unless the task set is found unschedulable in the meanwhile. The pseudo-code of the algorithm follows.

```
τ = {τ1, ..., τn}
begin
   for j = n downto 1
        unassigned = TRUE;
        for all tasks τk in τ
             if τk is feasible at priority j then
                  assign τk to priority j;
                  τ = τ − {τk};
                  unassigned = FALSE;
             end if
             if unassigned= TRUE then
                  exit; /* no priority assignment exists for τ
             end if
        end for
   end for
end
```

Clearly, as for the preemptive case [AUD91], the time complexity of this procedure is $O(n^2)$.

5. Shared resources and release jitter

The above analyses assume that all tasks do not share resources. In real operating systems, however, this needs not be the case. We must therefore extend the analysis to deal with priority inversion problems when dealing with dependencies. Furthermore, and for reasons such as tick scheduling or distributed context (e.g. the holistic approach introduced by [TIN95] for fixed priority scheduling, extended for dynamic priority scheduling in [SPU96-2]), tasks may be allowed to have a release jitter. In this section, we give some hints on how the analysis described previously must be modified in order to extend the model accordingly. The interested reader may refer to the given references.

5.1 Dynamic priority driven schedulers

5.1.1 Preemptive case

If the tasks are allowed to share resources, the analysis must take into account additional terms, namely blocking factors, owning to inevitable priority inversions. Note that:

- the maximum duration of such inversions can be bounded if shared resources are accessed by locking and unlocking semaphores according to a protocol like the priority ceiling [CL90], [SRL90] or the stack resource policy [BAK91]. In particular, for each task τ_i it is possible to compute the worst-case blocking time B_i, the maximum time a task τ_i may be blocked by lower priority tasks when accessing a shared resource.

- the length L of the processor busy periods is unaffected by the presence of blocking instead. Priority inversions may only cause the schedule to deviate from its ordinary EDF characteristic. The required modifications on the analysis are only few. The instance being checked, or another one which precedes it in the schedule, may experience a blocking that has to be include as an additional term.

On the other hand, if a task τ_i is delayed for a maximum time J_i (its release jitter) before being actually released, then two consecutive instances of τ_i may be separated by the interval $T_i - J_i$.

[SPU96] examines the feasibility of a general task sets in presence of shared resources and release jitter.

Theorem 18 - *([SPU96]) a general task set in presence of shared resources and release jitters is feasible (assuming that tasks are ordered by increasing value of D_i-J_i), using EDF, if:*

$$\forall t \le L, \qquad \sum_{D_i \le t + J_i} \left(1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \right) C_i + B_{k(t)} \le t \quad . \tag{10}$$

where L is the size of the synchronous processor busy period (see annex A) and $k(t) = max\{k | (D_k - J_k \le t)\}$.

The proof generalizes Theorem 6 showing that the worst processor busy period is still the synchronous processor busy period and that the worst pattern of arrival, considering release jitter, is when tasks experience their shortest inter-release times at the beginning of the schedule. Considering shared resources, it is shown that the worst pattern of arrival arises with the blocking factors of the task with the largest $D_k - J_k$ value among those included in the sum. Similarly, [SPU96] develops the same arguments for the computation of the worst-case response time.

As done in Section 3.1 the efficiency of the feasibility analysis and the worst-case response time computation can be improved by using the concept of deadline busy period.

5.1.2 Non-preemptive case

The feasibility analysis of a general task set in non-preemptive context (see Theorem 14) or in presence of blocking factor (see Theorem 18) are quite similar. This is not surprising since both refer to priority inversions w.r.t. absolute deadlines. It is also interesting to notice that resource sharing is quite simple to manage in non-preemptive context since, owing to the absence of preemption, there is no need of any particular protocol. The only possible priority inversions are caused by the absence of preemption. On the other hand, the worst patterns of arrival, when release jitter is considered, are similar to those identified in preemptive context, i.e., when tasks experience their shortest inter-release times at the beginning of the schedule. Therefore, to deal with the context of this section, Equation (10) in Theorem 18 only has to be replaced by:

$$\forall t \le L, \quad \sum_{D_i \le t + J_i} \left(1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \right) C_i + max_{D_i - J_i > t} \{ C_i - 1 \} \le t$$

with the convention that $max_{D_i - J_i > t} \{ C_i - 1 \} = 0$ *if* $\nexists i$*:* $D_i - J_i > t$.

5.2 Fixed priority driven schedulers

5.2.1 Preemptive case

Taking into account shared resources and release jitter in presence of fixed priority driven scheduling leads to the same reasoning than in presence of dynamic priority driven scheduling.

First, when $T_i = D_i$, $\forall i \in [1, n]$ Sha, Rajkumar and Lehoczky [SRL90] have extended the sufficient condition of [LL73] (see Theorem 8) for the Priority Ceiling Protocol:

$$\sum_{j=1}^{n} \frac{C_j}{T_j} + \frac{B_i}{T_i} \le i(2^{1/i} - 1), \; 1 \le i \le n$$

where B_i denotes the longest blocking time of τ_i a lower priority task (see Section 5.1.1).

The same enhancement can also be applied to the Lehoczky's busy period analysis. Moreover [TBW94], in presence of general task sets, extends this analysis further by considering the notion of release jitter (J_i for τ_i). Their analysis is an extension of Theorem 10 that results in the following final condition:

Theorem 19 - *([TBW94]) The worst-case response time* r_i *of a task* τ_i *of a general task set in presence of shared resources and release jitters is found in a scenario in which all tasks are at their maximum rate and released synchronously at a critical instant t=0.* r_i *is computed by the following recursive equation (where hp(i) denotes the set of tasks of higher priority than task* τ_i*):*

$$r_i = max_q (w_{i,q} + J_i - qT_i) \tag{11}$$

$$where\ w_{i,q} = (q+1)C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_{i,q} + J_j}{T_j} \right\rceil C_j \quad (12)$$

Furthermore, the general task set is feasible if: $\forall i \in [1, n]$, $r_i \leq D_i$.

5.2.2 Non-preemptive case

As for the non-preemptive dynamic case, resource sharing is quite simple to manage in non-preemptive fixed priority context since, owing to the absence of preemption, there is still no need of any particular protocol like the priority ceiling. The only possible priority inversions are then caused by the absence of preemption (see Theorem 15). On the other hand, the worst patterns of arrival, considering release jitter in non-preemptive context, are also similar to those identify in preemptive context, i.e., when tasks experience their shortest inter-release times at the beginning of the schedule. Therefore, to deal with the context of this section, Equation (11) and Equation (12) only have to be replaced by:

$$r_i = max_q(w_{i,q} + C_i + J_i - qT_i) \quad (13)$$

$$where\ w_{i,q} = qC_i + \sum_{j \in hp(i)} \left(1 + \left\lfloor \frac{w_{i,q} + J_j}{T_j} \right\rfloor \right) C_j + max_{k \in lp(i)}\{C_k - 1\}. \quad (14)$$

Note that a similar approach has been described in [TBW95] for the analysis of real-time networks in which packetized messages have access to the physical medium according to their fixed priorities.

6. Synthesis

As stated in Section 2.3, the goal of this paper was to fill in Table 1 for general task sets (i.e., to put together optimality properties, feasibility condition & worst-case response times in non-idling, preemptive/non-preemptive, fixed/dynamic priority driven contexts). To that end, we can now summarize the foregoing in Table 2 where white cells denote existing results when grey cells denote an extension of existing results or new results.

TABLE 2. Results for general task sets

	Preemptive scheduling		Non-preemptive scheduling	
	Dynamic priorities Section 3.1	Fixed priorities Section 3.2	Dynamic priorities Section 4.2	Fixed priorities Section 4.3
Optimality	Theorem 1	Theorem 7	Theorem 11	Section 4.3.2
Feasibility conditions	Theorem 6	Theorem 10	Theorem 14	Theorem 15
Worst-case Response times	Section 3.1.3	Theorem 10	Section 4.2.3	Theorem 15

7. Conclusion

In this paper, we have focused on the scheduling of general task sets (i.e., non-concrete periodic or sporadic task set such as $\forall i \in [1, n]$, T_i and D_i are not related) as a central figure for the description of possible processor loads, in several contexts. Although a lot of results were already known, some new results have been established either in preemptive or in non preemptive context. Our hope is that, given a real-time problem, this work might be helpful to pick a solution from the plethora of results available. In particular, the optimality of preemptive/non-preemptive, fixed/dynamic priority driven scheduling algorithms, the respective feasibility conditions and worst-case response times have been examined (see Section 6 for a synoptic). Some classic extensions such as jitter and resource sharing have also been considered.

Although this work is not oriented toward a comparison of these results, it appears that preemptive and non-preemptive scheduling are closely related. Indeed, the optimal scheduling algorithms are similar in both cases. The main differences, owning to the absence of preemption, are first that a task instance with lower priority can possibly cause a priority inversion before a higher priority busy period; and second that when estimating the response time of a task instance, the attention must be on the higher priority busy period preceding the execution start time of the instance, and not on the higher priority busy period preceding its completion time. These differences slightly change the feasibility conditions and the worst-case response time expressions but the whole analysis is very similar.

Moreover, fixed and dynamic scheduling differ, as already known, in many aspects such as implementability, efficiency, complexity, etc.. This is not surprising given that the optimality property of EDF is more general than with any fixed priority scheduler (the interested reader is referred to [HLR96] for a formal comparison of the efficiency and the complexity of fixed/dynamic priority driven scheduling in preemptive context). However, the analysis can be unified by using the concepts of processor busy period, that are scheduler independent, as well as higher priority busy period, that are scheduler dependent. Indeed, it appears that these concepts are general and very useful for the identification of the worst possible density of arrival and worst-case response times. In particular, we have introduced the concept of deadline-*d* busy period for dynamic priority driven scheduling that we conjecture as an interesting parallel of the level-*i* busy period already used in fixed priority driven scheduling. The main differences we detect are:

- first that the impact of the priority inversions caused by the absence of preemption disappear swiftly using EDF (after $max\{D_i\}$ in any deadline-*d* busy period) when it persists throughout the level-*i* busy period in the fixed priority case. This remark is in favour of EDF when preemption is not allowed, and is not surprising since any priority inversion, using EDF, refers to an absolute deadline (and not a fixed priority level) that is dynamically managed.

- second that any general task set which is feasible by EDF in non-preemptive context is necessarily feasible with EDF in preemptive context. This property doesn't hold when fixed priority assignment are considered (e.g., in Section 4.3.2. a task set that is feasible in non-preemptive context but not feasible in preemptive context is given).
  Hence conversely to EDF priority assignment, there's no obvious relationship between the feasibility of a fixed priority assignment in preemptive and non-preemptive contexts,

The main question which still remains open is whether there exists a fully polynomial solution to the feasibility problem for general task sets. Even if notions like processor demand, deadline busy period and level-*i* busy period have been helpful for the improvements of the known solutions, both in the dynamic and in the fixed priorities models the state of the art is represented by pseudo-polynomial algorithms. Whether the problem is NP-hard is also not known at present [BHR90].

References

[AUD91] N. C. Audsley, “Optimal priority assignment and Feasibility of static priority tasks with arbitrary start times”, Dept. Comp. Science Report YCS 164, University of York, 1991.

[BAK91] T.P. Baker, “Stack-Based Scheduling of Real-Time Processes,” Real-Time Systems, 3, pp. 67-99, 1991.

[BHR90] S. K. Baruah, R. R. Howell, L. E. Rosier, “Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time tasks on one processor”, Real-Time Systems, 2 , pp. 301-324, 1990.

[BHR93] S. Baruah, R. Howell, and L. Rosier. “Feasibility Problems for Recurring Tasks on One Processor”. Theoret. Comput. Sci. 118 (1993), pp. 3-20. (84 K).

[BMR90] S. K. Baruah, A. K. Mok, L. E. Rosier, “Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor”, Proceedings of the 11th Real-Time Systems Symposium, p 182-190, 1990.

[CL90] M. Chen and K. Lin, “Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems,” Real-Time Systems, 2, pp. 325-345, 1990.

[COF76] E.G. Coffman, Jr., “Introduction to Deterministic Scheduling Theory,” in: E.G. Coffman, Jr., Ed., Computer and Job-Shop Scheduling Theory, Wiley, New York, 1976.

[DER74] M. Dertouzos, “Control Robotics: the procedural control of physical processors”, Proceedings of the IFIP congress, p 807-813, 1974.

[GJ79] M. R. Garey, D. S. Johnson, “Computer and Intractability, a Guide to the Theory of NP-Completeness”, W. H. Freeman Company, San Francisco, 1979.

[GMR95] L.George, P.Muhlethaler, N. Rivierre, “Optimality and Non-Preemptive Real-Time Scheduling Revisited,” Rapport de Recherche RR-2516, INRIA, Le Chesnay Cedex, France, 1995.

[HLR96] J.F. Hermant, L. Lebouche, N. Rivierre, “On n comparing fixed/dynamic priority driven scheduling algorithms” Rapport de Recherche, INRIA, Le Chesnay Cedex, France, to appear in 1996.

[HV95] R. R. Howell, M. K. Venkatrao, “On non-preemptive scheduling of recurring tasks using inserted idle time”, Information and computation Journal, Vol. 117, Number 1, Feb. 15, 1995.

[JP86] M. Joseph, P. Pandya, “Finding response times in a real-time system”, BCS Comp. Jour., 29(5), pp. 390-395, 1986.

[JSM91] 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.

[KLS93] D. I. Katcher, J. P. Lehoczky, J. K. Strosnider, “Scheduling models of dynamic priority schedulers”, Research Report CMUCDS-93-4, Carnegie Mellon University, Pittsburgh, April 1993.

[KN80] Kim, Naghibdadeh, “Prevention of task overruns in real-time non-preemptive multiprogramming systems”, Proc. of Perf., Assoc. Comp. Mach., 1980, pp 267-276,

[LEH90] J.P. Lehoczky, “Fixed priority scheduling of periodic task sets with arbitrary deadlines”, Proceedings 11th IEEE Real-Time Systems Symposium, Lake Buena Vista, FL, USA, pp. 201-209, 5-7 Dec. 1990.

[LL73] 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, No 1, Janv 1973.

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

[LS95] L. Leboucher, Jean-Bernard Stefani, “Admission control for end-to-end distributed bindings“, COST231, Lectures Notes in Computer Science, Vol 1052, pp192,208, Nov. 1995.

[LW82] J. Leung, J. Whitehead. “On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation, vol 2, p237-250, 1982.

[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.

[RCM96] I. Ripoll, A. Crespo, A.K. Mok, “Improvement in Feasibility Testing for Real-Time Tasks,” Real-Time Systems, 11, 1996, pp 19-39.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky, “Priority Inheritance Protocols: An Approach to Real-Time Synchronization,” IEEE Transactions on Computers, 39(9), 1990.

[SPU95] M. Spuri, “Earliest Deadline scheduling in real-time systems”, Doctorate dissertation, Scuola Superiore S. Anna, Pisa, 1995.

[SPU96] M. Spuri, “Analysis of deadline scheduled real-time systems” Research Report 2772, INRIA, France, Jan. 1996.

[SPU96-2] M. Spuri, “Holistic analysis for deadline scheduled real-time distributed systems” Research Report 2873, INRIA, France, Apr. 1996.

[STA95] J.A. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo, “Implications of Classical Scheduling Results for Real-Time Systems,” IEEE Computer, June 1995.

[TBW94] K. Tindell, A. Burns, A. J. Wellings, “An extendible approach for analyzing fixed priority hard real time tasks”, Real-Time Systems, 6, 1994, p133-151.

[TBW95] K. Tindell, A. Burns, A. J. Wellings, “Analysis of hard real-time communications”, Real-Time Systems, 9, 1995, p147-171.

[THW94] K. Tindell, H. Hansson, A. J. Wellings, “analyzing real-time communications: controller Area Networks (CAN)”, Real-Time Systems, 1994, p259-263.

[TIN95] K. Tindell, “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems“, Euromicro Journal, Special Issue on Embedded Real-Time Systems, Feb. 1995.

[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).

[ZRS87] 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.

[ZS94] Q. Zheng, K.G. Shin, “On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks,” IEEE Transactions on Communications, 42(2/3/4), 1994.

Annexe A. Busy Periods, Definitions and Properties

The notion of processor busy period is very simple, but at the same time also very powerful, since the properties (feasibility and worst-case response times) of fixed and dynamic priority non-idling schedulers can be exactly characterized by using this concept. More precisely, we first focus on the notion of processor busy period that does not depend on the scheduling algorithm, and then on the notion of "higher priority" busy period, that is scheduling algorithm dependent.

A.1 Processor Busy Periods

DEFINITION 1 - A *processor busy period* is an interval of time in which the processor is kept continually busy by the execution of pending instances.

Note that in the definition nothing is said about what delimits the interval at its sides. However, unless otherwise stated we will usually intend a time interval delimited by two distinct processor idle periods, i.e., any periods such that no outstanding computation exists.[1]

Among all processor busy periods, we find particularly interesting the "first" one obtained by releasing all tasks synchronously from time $t = 0$.

DEFINITION 2 - Given a general task set, we call *synchronous busy period* the processor busy period beginning at time $t = 0$ and delimited by the first processor idle period, when all tasks are concretely released from time $t = 0$ on at their maximum rate.

The importance of the synchronous busy period is that, not surprisingly enough, it is the most demanding one for most of the non-idling scheduling algorithms. Also, we can easily prove that is the largest possible busy period.

Lemma 8 - *Given a general task set, let L be the length of its synchronous busy period. If L' is the length of a processor busy period in the schedule of any derived concrete task set, then*

$$L' \leq L.$$

Proof. The given processor busy period must be preceded by an idle time, and its beginning must coincide with the release time *t* of a task instance. If all instances of any task released after *t* are shifted left as much as possible, possibly up to *t*, we obtain a synchronous busy period starting at *t*. Since the workload between *t* and $t + L'$ cannot decrease with the shift-left argument, the length of the new busy period cannot diminish. Hence, $L' \leq L$. ❑

Note that the property established by the previous lemma does not depend on the scheduling algorithm assumed, either static or dynamic, preemptive or non-preemptive. The only assumption is that the algorithm is non-idling.

The length L of the synchronous busy period can be computed by means of a simple procedure. Given any interval $[0, t[$, the idea is to compare the generated workload $W(t)$ with the length t of the interval: if $W(t)$ is greater than t then the duration of the busy period is at least $W(t)$. The argument is

1. Note that a processor idle period can have zero duration, when a new task instance is released at the end of a busy period.

then recursively applied to $W(t)$, $W(W(t))$, … , until we find a value equal to the previous one. Formally, L is the fixed point of the following iterative computation:

$$\begin{cases} L^{(0)} = \sum_i C_i \\ L^{(m+1)} = W(L^{(m)}). \end{cases} \tag{15}$$

The computation is halted when two consecutive values $L^{(m)}$ and $L^{(m+1)}$ are found equal. L is then assigned the value $L^{(m)}$. Note that in this way we find the smallest solution of the equation

$$L = W(L). \tag{16}$$

The convergence of Equation (15) is proved in the following lemma.

Lemma 9 - *If $\sum_i C_i/T_i \le 1$, Equation (15) converges in a finite number of steps.*

Proof.

$$\sum_i \frac{C_i}{T_i} \le 1 \quad \Rightarrow \quad W(P) = \sum_i \left\lceil \frac{P}{T_i} \right\rceil C_i = P \sum_i \frac{C_i}{T_i} \le P.$$

It follows that $L \le P$. Furthermore, the workload function $W(t)$ is a non-decreasing step-function, hence $L^{(m)}$ is non-decreasing in m. Finally, at each step $L^{(m)}$ is either increased by at least C_{min} or remains unchanged. Thus the final value is reached in a finite number of steps. ❑

If we have the further hypothesis that $\sum_i C_i/T_i \le c$, where c is a constant smaller than 1, then the complexity of L's computation becomes pseudo-polynomial [SPU96]. To show this, we just need few algebraic manipulations:

$$L = \sum_i \left\lceil \frac{L}{T_i} \right\rceil C_i \le \sum_i \left(1 + \frac{L}{T_i} \right) C_i = \sum_i C_i + L \sum_i \frac{C_i}{T_i} \le \sum_i C_i + cL.$$

Hence

$$L \le \frac{\sum_i C_i}{1 - c}.$$

Each step of the iterative formula Equation (15) takes $O(n)$ time, thus the whole computation takes $O(n(\sum_i C_i)/C_{min})$ time.[1]

1. Note that finding a full polynomial time algorithm for the computation of L would imply a full polynomial time procedure for the feasibility assessment of a general task set (see section 3.1.2), which is still an open question [BHR90].

A.2 "Higher Priority" Busy Periods

So far, the concept of busy period has been introduced as such, that is, as a period in which the processor is not idle. More generally, it is possible to specialize the concept by making it relative to a given level of priority. The idea is to consider processor busy periods in which only task instances with priority, either static or dynamic, at least at a given level are executed. This form of busy period is scheduling algorithm dependent, and in particular it is useful for the computation of tasks worst-case response times.

In the case of fixed priority systems, we talk of *level-i busy periods* [LEH90]:

DEFINITION 3 - A *level-i busy period* is a processor busy period in which only instances of tasks with priority greater than or equal to that of τ_i execute**.**

For the computation of worst-case response times it is necessary to compute the length of suitable level-*i* busy periods. The approach is very similar to that seen for the computation of the synchronous busy period length. In this case, though, only the workload of the tasks with priority greater than or equal to that of τ_i is taken into account.

The length of the longest level-i busy period is denoted by L_i, and can be easily computed by finding the smallest solution of the equation

$$L_i = \sum_{j \in hp(i) \cup \{i\}} \left\lceil \frac{L_i}{T_j} \right\rceil C_j,$$

for the preemptive case, and

$$L_i = max_{j \in lp(i)}\{C_j - 1\} + \sum_{j \in hp(i) \cup \{i\}} \left\lceil \frac{L_i}{T_j} \right\rceil C_j,$$

for the non-preemptive case, respectively. Note that in the latter case, the first term on the right-hand size takes into account the effect of non-preemption, that is, the non-preemptable execution of a lower priority task instance can delay the executions of higher priority task instances as much as $max_{j \in lp(i)}\{C_j - 1\}$. As usual the equations can be solved by means of iterative computations.

Similarly, for dynamic priority systems, and in particular for deadline scheduled ones, we talk of *deadline busy periods:*

DEFINITION 4 - A *deadline-d busy period* is a processor busy period in which only task instances with absolute deadline smaller than or equal to d execute.

Again, the approach for the computation of deadline-d busy period lengths is as usual. We only need to be careful in taking into account the right number of instances for each task, which requires slightly more changes. In fact, in this case given an interval of time $[0, t)$, the number of instances of τ_i released within t is $\lceil t/T_i \rceil$, however only $1 + \lfloor (d - D_i)/T_i \rfloor$, provided that $D_i \leq d$, 0 otherwise,

have an absolute deadline smaller than or equal to d. Consequently, the number of τ_i 's instances to be taken into account is

$$min\left\{\left\lceil \frac{t}{T_i} \right\rceil, 1 + \left\lfloor \frac{d - D_i}{T_i} \right\rfloor\right\},$$

which gives a sort of "higher priority workload."

Also, in this case we are not merely interested in any deadline-d busy period, but only on those which include the last instance. More precisely, given a τ_i 's instance released at time a, hence with deadline $d = a + D_i$ (see figure 8), we are interested in deadline-d busy periods that starts at $t = 0$ and includes this instance (i.e., when its length is bigger than a [1]). It is not difficult to see that the longest busy periods are obtained in scenarios in which all tasks but τ_i are released synchronously (see Lemma 3).

Let $L_i(a)$ denotes the length of such deadline-d busy period that ends on the completion time of the τ_i 's instance released at time a (with $d = a + D_i$), while all other tasks are considered to be released synchronously ($s_j = 0, \forall j \neq i$). The deadline busy periods in which we are interested are only those for which

$$L_i(a) > a.$$

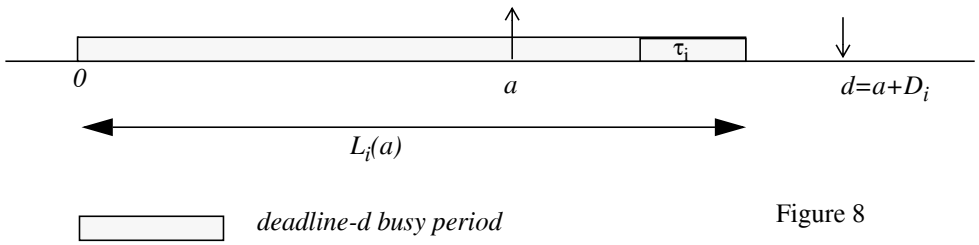


Figure 8

$L_i(a)$ is then computed by means of the iterative computation

$$L_i^{(0)}(a) = 0, \quad L_i^{(m+1)}(a) = W_i(a, L_i^{(m)}(a)) + (1 + \lfloor a/T_i \rfloor)C_i,$$

where

$$W_i(a, t) = \sum_{\substack{D_j \leq a + D_i \\ j \neq i}} min\left\{\left\lceil \frac{t}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right\}C_j.$$

The computation is halted when two consecutive values are found equal. Note that in this way we find the smallest solution of the equation

1. Note that if the deadline-d busy period is shorter than a, then it is not interesting neither for the feasibility checking (according to Lemma 1 the deadline d cannot be missed), nor for the computation of the worst-case response time of τ_i (see Section 3.1.3).

$$t = W_i(a, t) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i. \tag{17}$$

With L_i we denote the length of the longest such deadline busy period for task τ_i. We have then

$$L_i = max_{a < L_i(a)}\{L_i(a)\}.$$

In order to speed up the computation of L_i, we can utilize two nice properties shown in the following lemma.

Lemma 10 - $D_i \leq D_j \Rightarrow L_i \leq L_j$

Proof. By definition of L_i, $\exists d = s_i + mT_i + D_i$ for which $L_i = L_i(d - D_i)$:

$$\begin{aligned} L_i &= \sum_{\substack{D_k \leq d \\ k \neq i}} min\left\{\left\lceil \frac{L_i}{T_k} \right\rceil, 1 + \left\lfloor \frac{d - D_k}{T_k} \right\rfloor\right\} C_k + (m+1)C_i \\ &= \sum_{\substack{D_k \leq d \\ k \neq i \\ k \neq j}} min\left\{\left\lceil \frac{L_i}{T_k} \right\rceil, 1 + \left\lfloor \frac{d - D_k}{T_k} \right\rfloor\right\} C_k + (m+1)C_i + min\left\{\left\lceil \frac{L_i}{T_j} \right\rceil, 1 + \left\lfloor \frac{d - D_j}{T_j} \right\rfloor\right\} C_j. \end{aligned}$$

Since $D_i \leq D_j$, the last τ_j 's instance with deadline before or at d has release time smaller than L_i,[1] then

$$L_i = \sum_{\substack{D_k \leq d \\ k \neq i \\ k \neq j}} min\left\{\left\lceil \frac{L_i}{T_k} \right\rceil, 1 + \left\lfloor \frac{d - D_k}{T_k} \right\rfloor\right\} C_k + (m+1)C_i + \left(1 + \left\lfloor \frac{d - D_j}{T_j} \right\rfloor\right) C_j,$$

If we consider now the scenario in which τ_i is released synchronously (i.e., s_i becomes 0) and τ_j has start time $s_j = (d - D_j) - \lfloor (d - D_j)/T_j \rfloor T_j$, so that there is an instance with deadline d, we have

$$L_i \leq \sum_{\substack{D_k \leq d \\ k \neq j}} min\left\{\left\lceil \frac{L_i}{T_k} \right\rceil, 1 + \left\lfloor \frac{d - D_k}{T_k} \right\rfloor\right\} C_k + \left(1 + \left\lfloor \frac{d - D_j}{T_j} \right\rfloor\right) C_j.$$

The right quantity of the inequality is smaller than or equal to the new deadline-d busy period, which in turn is by definition smaller than or equal to L_j. The thesis follows. ❑

1. We assume $D_j \leq d$. If this is not true, the thesis trivially holds.

Lemma 11 - $\forall i, L_i(a)$ is non-decreasing in a.

Proof. By definition, $L_i(a)$ is the smallest solution of Equation (17), hence for any $t < L_i(a)$

$$t < W_i(a, t) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i.$$

Given $a' \geq a$, we thus that $\forall t < L_i(a)$

$$t < W_i(a, t) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i \leq W_i(a', t) + \left(1 + \left\lfloor \frac{a'}{T_i} \right\rfloor\right) C_i,$$

since $W_i(a, t)$ is non-decreasing in a. It follows that $L_i(a') \geq L_i(a)$. ❑

By using the results of Lemma 8, Lemma 10 and Lemma 11, we can compute the values of L_i in the following way. Let $E = \bigcup_{i=1}^{n} \{mTi + D_i, m \geq 0\} = \{e_1, e_2, \ldots\}$, let L be the length of the synchronous busy period, and assume that $D_i \leq D_{i+1}, \forall i$. The algorithm for the computation of L_i starts by placing an instance of τ_i in such a way to have its deadline at the largest e_k smaller than or equal to $L_{i+1} - C_i + D_i$ (it is easy to see that these are the scenarios which give the longest deadline busy periods for τ_i). If the resulting deadline-e_k busy period includes this last instance of τ_i we have found L_i, otherwise the computation is done again by choosing a new suitable deadline $e_{k'}$, this time according to the value of $L_i(e_k - D_i)$. Note that the algorithm always stops. The pseudo-code follows:

$L_{n+1} = L$;
for i=n **downto** 1
    let k be such that $e_k \leq L_{i+1} - C_i + D_i < e_{k+1}$;
    a = $e_k - D_i$;
    while $L_i(a) \leq a$
        let k be such that $e_k \leq L_i(a) - C_i + D_i < e_{k+1}$;
        a = $e_k - D_i$;
    end while
    $L_i = L_i(a)$;
end for

Note that the algorithm is also valid when preemption is not allowed. In this case, however, the equation for the computation of $L_i(a)$ is slightly different, since we must take into account the effect of non-preemption, namely, possible priority inversions. In particular, the equation becomes:

$$L_i(a) = B + \sum_{\substack{D_k \le a + D_i \\ k \ne i}} min\left\{\left\lceil \frac{L_i(a)}{T_k} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_k}{T_k} \right\rfloor\right\} C_k + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i,$$

and

$$B = max_{D_j > a + D_i}\{C_j - 1\}.$$

If our goal is only to check the feasibility of the task set (and not to do a full worst-case response times analysis), the value of L_i computed by the previous algorithm might be unnecessarily large. This is because when checking the feasibility, by Lemma 1 we can focus our attention only on synchronous deadline busy periods ($s_i = 0, \forall i$). Unfortunately, in this way we loose the property shown in Lemma 10, since we cannot apply the same "swapping" argument. For any task τ_i, the new value L_i^s is then computed in the following way:

a = $\lceil (L - D_i)/T_i \rceil T_i$;
while $L_i(a) \le a$
    a = $\lceil (L_i(a) - D_i)/T_i \rceil T_i$;
end while
$L_i^s = L_i(a)$;

Other details of the "higher priority" busy period length computations, either for static or for dynamic priority models, are given in the specific sections, where the differences between the preemptive and the non-preemptive models (actually very few) are also remarked.

Annexe B. Polynomial sufficient conditions (SC)

Let us now introduce in preemptive context some possible polynomial sufficient, but not necessary, feasibility conditions for general task sets. The main objective of this annex is to propose several approaches that will enable us to avoid the cost of the pseudo-polynomial NSC when possible.

B.1 Sufficient Conditions for dynamic priority driven preemptive schedulers

Using EDF, the first SC is established, using classic overvaluation of $h(t)$ and deriving an upper bound on the authorized processor utilization (see Theorem 20). The second SC is established by an overvaluation of $\frac{h(t)}{t}$ enabling us to check this expression on exactly n worst-case points (see Theorem 21). A third and fourth SC will establish upper-bounds on the worst-case response times, based on the synchronous pattern of arrival only (and not those defined in Section 3.1.3). More precisely, the third SC uses overvaluation on $W_i(a,t)$ that enables us to avoid the cost of the recursive analysis (see Theorem 22). Then, the fourth SC uses an extra overvaluation that enables us to obtain a polynomial computation time upper bound on the worst-case response time of a task (see Theorem 22).

Theorem 20 - *A general task set (with $D'_i = min(D_i, T_i) \quad \forall i \in [1, n]$) is feasible, using EDF, if:*

$$U \le 1 - \frac{1}{min(D_i)} \sum_{i=1}^{n} \left(1 - \frac{D'_i}{T_i}\right) C_i \tag{18}$$

Proof. from the state of the art, a NSC is: $\forall t \ge 0 \qquad h(t) = \sum_{D_i \le t} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i \le t$

As $\forall i \in [1, n]$, $D'_i \le T_i$, eliminating the floor function and by the definition of *U*.

$$\forall t \ge 0 \qquad h(t) \le \sum_{D_i \le t} \left(\frac{t + T_i - D_i}{T_i}\right) C_i \le \sum_{D_i \le t} \left(\frac{t + T_i - D'_i}{T_i}\right) C_i \le t \cdot U + \sum_{i=1}^{n} \left(1 - \frac{D'_i}{T_i}\right) C_i$$

Therefore a sufficient feasibility condition is to check that $\forall t \ge 0$, $t \cdot U + \sum_{i=1}^{n} \left(1 - \frac{D'_i}{T_i}\right) C_i \le t$ i.e

$U \le 1 - \frac{1}{t} \sum_{i=1}^{n} \left(1 - \frac{D'_i}{T_i}\right) C_i$. As it is sufficient to check the feasibility on the absolute deadlines and as

$1 - \frac{1}{t} \sum_{i=1}^{n} \left(1 - \frac{D'_i}{T_i}\right) C_i$ is an increasing function by t, a least upper bound on the processor utilization is obtained at $t = min(D_i)$. Hence a feasibility condition is to check that

$$U \le 1 - \frac{1}{min(D_i)} \sum_{i=1}^{n} \left(1 - \frac{D'_i}{T_i}\right) C_i . \qquad \square$$

Theorem 21 - *A general task set sorted by increasing order of* $D'_i = min(D_i, T_i)$ *is feasible using EDF if:* $\forall i \in [1, n]$, $\sum_{D_j \le D_i} \left(\frac{D_i + T_j - D_j'}{T_j} \right) C_j \le D_i$

Proof. Let $h_i(t)$ be the processor demand for task τ_i, if $t \ge D_i$ we have $\frac{h_i(t)}{t} = \frac{1}{t}\left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i$ and $\frac{h_i(t)}{t} = 0$ else. $\forall t \ge D_i$, we have:

$$\frac{h_i(t)}{t} = \frac{1}{t}\left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) C_i \le \frac{1}{t}\left(\frac{t + T_i - D_i}{T_i} \right) C_i \le \frac{1}{t}\left(\frac{t + T_i - D_i'}{T_i} \right) C_i .$$

In the same manner, we have:

$$\frac{h(t)}{t} = \sum_{j=1}^{n} \frac{h_j(t)}{t} = \frac{1}{t} \sum_{D_j \le t} \left(1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor \right) C_j \le \frac{1}{t} \sum_{D_j \le t} \left(\frac{t + T_j - D_j'}{T_j} \right) C_j .$$

As $\frac{1}{t}\left(\frac{t + T_i - D_i'}{T_i} \right) C_i$ is a non decreasing function, its maximum value is obtained for $t = D_i$. i.e.

$$\frac{h_i(t)}{t} \le \frac{1}{D_i}\left(\frac{D_i + T_i - D_i'}{T_i} \right) C_i .$$

Hence $\forall i \in [1, n]$, $\forall t \in [D_i, D_{i+1})$,

$$\frac{h(t)}{t} \le \frac{1}{D_i} \sum_{D_j \le D_i} \left(\frac{D_i + T_j - D_j'}{T_j} \right) C_j . \qquad (19)$$

What's more, $\forall t \ge max(D_i)$,

$$\frac{h(t)}{t} \le \frac{1}{t} \sum_j \left(\frac{t + T_j - D_j'}{T_j} \right) C_j \le \frac{1}{max(D_i)} \sum_j \left(\frac{max(D_i) + T_j - D_j'}{T_j} \right) C_j \quad . \qquad (20)$$

As a NSC for the feasibility of a task set is to check that $\forall t \ge 0$, $h(t) \le t$ (see Theorem 6). Applying this to eq. 19 and eq. 20, a sufficient feasibility condition is to check that $\forall i \in [1, n]$ **,**

$$\sum_{D_j \le D_i} \left(\frac{D_i + T_j - D_j'}{T_j} \right) C_j \le D_i \qquad \square$$

Theorem 22 - *Using EDF, an upper bound of the worst-case response time of a task τ_i can be found setting the absolute deadline of τ_i on every absolute deadlines of the tasks in the synchronous busy period and is given by:*

$$\max_{a \in S}\left(\sum_{D_j \le a + D_i}\left(1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right)C_j - a\right).$$

Where $S = \bigcup_{p=1}^{n} S_p$, $S_p = \left\{D_p + kT_p - D_i, k \in \aleph, \left\lceil \frac{D_i - D_p}{T_p} \right\rceil \le k < \left\lceil \frac{L + D_i - D_p}{T_p} \right\rceil\right\}$ *and* L *is the length of the synchronous processor busy period (see Annex A).*

Proof. Let τ_i be a task released at time t. The higher priority workload (see Section 3.1.3) arrived up to time t can be reformulated as

$$W_i(a,t) = \sum_{D_j \le a + D_i} min\left(\left\lceil \frac{t - s_j}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right)C_j.$$

Where $s_j = 0$ if $j \neq i$ and $s_j = a - \left\lfloor \frac{a}{T_i} \right\rfloor T_i$ else.

The worst-case response time of task τ_i is then given by $r_i(a) = max(C_i, L_i(a) - a)$ where $L_i(a) = W(a, L_i(a))$. $\forall t > 0$, we have then:

$$W_i(a,t) \le \sum_{D_j \le a + D_i}\left(1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right)C_j.$$

Hence, $L_i(a) \le \sum_{D_j \le a + D_i}\left(1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right)C_j$. As $C_i \le \sum_{D_j \le D_i}\left(1 + \left\lfloor \frac{D_i - D_j}{T_j} \right\rfloor\right)C_j$, we have:

$$r_i \le \max_{a \in [0,L)}\left(\sum_{D_j \le a + D_i}\left(1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right)C_j - a\right).$$

Let $f_i(a) = \sum_{D_j \le a + D_i}\left(1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right)C_j - a$. $f_i(a)$ is a step function whose value increase when $a + D_i - D_j = kT_j$ i.e such that the absolutes deadline of τ_i $(a + D_i)$ coincide to the absolute deadline of a task released in the synchronous scenario $(D_j + kT_j)$ leading to $a \in S_j$, $\forall j = 1 \ldots n$. Notice that $f_i(a)$ is monotonously decreasing between two successive values in S. Thus we only need to figure $f_i(a)$ for times in S. Hence, an upper-bound of the worst-case response time of task τ_i is then given by $\max_{a \in S}\left(\sum_{D_j \le a + D_i}\left(1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right)C_j - a\right)$. ❑

From the last formula, we can now give a polynomial time computation upper bound on the worst-case response time of any task τ_i. This bound is not tight but can be calculated on line.

Theorem 23 - *A upper bound r'_i of the worst-case response time of a task using EDF is* $r'_i = \sum_{D_j - T_j \le D_i} \left(\frac{D_i - D_j + T_j}{T_j} \right) C_j$.

Proof. From last theorem, we have: $r_i \le \max_{a \in S} \left(\sum_{D_j \le a + D_i} \left(1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right) C_j - a \right)$. Thus $\forall a \in S$, eliminating the floor function, we have:

$$r_i(a) \le \sum_{D_j \le a + D_i} \left(1 + \frac{a + D_i - D_j}{T_j} \right) C_j - a ,$$

that leads to:

$$r_i(a) \le \max \left(\sum_{D_j \le a + D_i} \left(\frac{D_i - D_j + T_j}{T_j} \right) C_j + \left(\sum_{D_j \le a + D_i} \frac{C_j}{T_j} - 1 \right) a \right).$$

As $\sum_{D_j \le a + D_i} \frac{C_j}{T_j} \le U \le 1$, we have $\forall a \in [0, L]$:

$$r_i(a) \le \sum_{D_j \le a + D_i} \left(\frac{D_i - D_j + T_j}{T_j} \right) C_j \le \sum_{D_j - T_j \le D_i} \left(\frac{D_i - D_j + T_j}{T_j} \right) C_j = r'_i . \qquad \square$$

B.2 Sufficient Condition for fixed priority driven preemptive schedulers

Using fixed priority driven schedulers, the following feasibility SC establishes upper-bounds on the worst-case response times. This result is established by first an overvaluation on $w_{i,q}$; and second by eliminating the term function of q.

Theorem 24 - *Using static schedulers, an upper bound of he worst-case response time r'_i of a task τ_i of a general task set is found by:*

$$r'_i = \left(\sum_{j \le i} C_j \right) / \left(1 - \sum_{j < i} \frac{C_j}{T_j} \right) \qquad (21)$$

Furthermore, the general task set is feasible if: $\forall i \in [1, n]$, $r'_i \le D_i$.

Proof. {from the state of the art, a NSC for static priority driven scheduler is (see Section 3.2.2)}

$$\forall (i, q) \qquad w_{i,q} = (q+1) C_i + \sum_{j < i} \left(\left\lceil \frac{w_{i,q}}{T_j} \right\rceil \right) C_j \le q T_i + D_i$$

{majoring the ceiling function}

$$w_{i,q} \le (q+1)C_i + \sum_{j<i}\left(\frac{w_{i,q}}{T_j}+1\right)C_j.$$

$$\Rightarrow w_{i,q} - qT_i \le \frac{(q+1)C_i + \sum_{j<i} C_j}{\left(1-\sum_{j<i}\frac{C_j}{T_j}\right)} - qT_i = \frac{qT_i\left(\sum_{j\le i}\frac{C_j}{T_j}-1\right)+\sum_{j\le i}C_j}{\left(1-\sum_{j<i}\frac{C_j}{T_j}\right)}$$

As $\sum_{j\le i}\frac{C_j}{T_j} \le U \le 1$

$$\Rightarrow r_i = max(w_{i,q} - qT_i) \le r'_i = \left(\sum_{j\le i} C_j\right) \Big/ \left(1-\sum_{j<i}\frac{C_j}{T_j}\right).$$ ❑

Unité de recherche INRIA Lorraine, technopôle de Nancy-Brabois, 615 rue du jardin botanique, BP 101, 54600 VILLERS-LÈS-NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, domaine de Voluceau, Rocquencourt, BP 105, LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
Inria, Domaine de Voluceau, Rocquencourt, BP 105 LE CHESNAY Cedex (France)

ISSN 0249-6399