INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Real-Time Scheduling: Non-Preemption, Critical Sections and Round Robin.

Jörn Martin Migge, Alain Jean-Marie

N° 3678

Avril 1999

THÈME 1

Rapport de recherche

ISSN 0249-6399

Real-Time Scheduling: Non-Preemption, Critical Sections and Round Robin.

Jörn Martin Migge, Alain Jean-Marie

Thème 1 — Réseaux et systèmes
Projet Mistral

Rapport de recherche n° 3678 — Avril 1999 — 67 pages

Abstract: The main property of a (hard) real-time system is feasibility. It is the guarantee that tasks do always meet their deadlines, when the system is running under a given scheduling policy. An optimal policy produces a feasible schedule whenever a feasible schedule exists. It is therefore the most appropriate choice if feasibility is the only matter of concern. However, if a set of tasks is feasible under several policies, it is possible to impose additional constraints, which improve in a certain way the quality of a system. A refined choice is only possible if feasibility tests or, more broadly, timing analysis is available for other policies than the optimal ones. In this document we derive timing analysis for policies obtained by combining known policies in hierarchical layers. These *layered priorities* are motivated by the Posix 1003.1c standard, which allows such a combination of Fixed Preemptive Priorities and the Round Robin scheduling policy. In this context we extend the trajectory based model developed in [8] for systems scheduled under real-time constraints to account for non-preemptive resources and the associated priority ceiling protocol. Furthermore, timing analysis of the Round Robin policy is derived.

Key-words: real-time scheduling, timing analysis, layered priorities, Round Robin

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Ordonnancement Temps Réel: Non-Préemption, Sections Critiques et Round Robin

Résumé : La propriété principale d'un système temps réel est la faisabilité. C'est la garantie que les tâches respectent toujours leurs dates d'échéance lorsque le système fonctionne avec une certaine politique d'ordonnancement. Une politique optimale produit un ordonnancement faisable si un ordonnancement faisable existe. Pour cette raison une politique optimale est la plus appropriée si la faisabilité est le seul objectif. Mais si un ensemble de tâches est faisable sous plusieurs politiques, il est possible d'ajouter d'autres contraintes que la faisabilité afin d'améliorer la qualité du système selon des critères négligés auparavant. Cependant, un choix plus raffiné n'est possible que si des analyses de faisabilité sont aussi connues pour d'autres politiques que les politiques optimales. Dans ce document nous développons des analyses de faisabilité pour des politiques d'ordonnancement obtenues par combinaison en couches hiérarchiques. Ces *priorités en couches* sont motivées par le standard POSIX 1003.1c, qui prévoit notamment la combinaison en couches de priorités préemptives fixes et Round Robin. Dans ce contexte nous étendons le modèle à base de trajectoires développé dans [8] pour les systèmes de tâches sous contraintes temps réels sur un processeur. Nous élaborons une représentation des ressources non-préemptables et du protocole à plafond de priorité associé. De plus, nous développons une analyse d'ordonnançabilité pour la politique Round Robin.

Mots-clés : ordonnancement, temps-réel, priorités en couches, Round Robin

Contents

1 Introduction 5

2 Framework 5
- 2.1 The task process . . . 5
  - 2.1.1 Recurrent tasks . . . 5
  - 2.1.2 Work arrival functions (WAF) . . . 7
- 2.2 The scheduled task process . . . 7
  - 2.2.1 The scheduling function . . . 8
  - 2.2.2 Response times . . . 9
  - 2.2.3 Feasibility . . . 10
- 2.3 Basic assumptions . . . 10
  - 2.3.1 Deterministic stability . . . 11
  - 2.3.2 Non-idling . . . 11
  - 2.3.3 Majorizing work arrival functions (MWAF) . . . 12
- 2.4 Highest Priority First Scheduling . . . 12
  - 2.4.1 Motivation and definition . . . 12
  - 2.4.2 Some formulas . . . 16
  - 2.4.3 Properties of priority assignments . . . 17

3 Time dependent priority assignments 19
- 3.1 Priority promotion . . . 19
  - 3.1.1 Priority functions with promotion at execution beginning (PPEB) . . . 19
  - 3.1.2 Function of demand for a resource . . . 24
  - 3.1.3 Priority Ceiling Protocol . . . 25
  - 3.1.4 Deadlocks and proof of existence . . . 31
  - 3.1.5 Locking periods . . . 32
- 3.2 Layered priorities . . . 32
  - 3.2.1 Layered preemptive priorities (LPP) . . . 33
  - 3.2.2 Priority promotion and ceilings under layered priorities . . . 33
- 3.3 Round Robin (RR) . . . 34
  - 3.3.1 Definition . . . 34
  - 3.3.2 Non-preemptable resources under Round Robin . . . 38
- 3.4 Non-preemptive policies . . . 39
  - 3.4.1 Non-preemptive versions of preemptive policies . . . 39
  - 3.4.2 Layered priorities and non-preemptive policies . . . 40
- 3.5 Posix 1003.1b compliant scheduling . . . 40
- 3.6 Proof of existence . . . 41

4 Response time bounds 43
- 4.1 Priority promotion at execution beginning . . . 43
  - 4.1.1 Generic bounds . . . 44
  - 4.1.2 Non-preemptive policies . . . 45
- 4.2 Response time bounds in the presence of non-preemptable resources . . . 48
  - 4.2.1 A generic Bound . . . 48
  - 4.2.2 Time independent priority functions in layers . . . 49
  - 4.2.3 Execution begin until execution end . . . 50
- 4.3 Round Robin . . . 50

5 Numerical applications **55**
5.1 Layered priorities . . . 55
5.2 Round Robin . . . 57
5.3 Non-preemptive policies in preemptive layers . . . 58
5.4 Bounds on blocking periods under PCP . . . 59

6 Conclusion **60**

A Miscellaneous properties **61**
A.1 Fixed point equations and iterative computation . . . 61
A.1.1 Fixed point . . . 61
A.1.2 Iterations . . . 62

List of Figures

1 Times and functions associated with the instance of a task τ_k. . . . 10
2 Two different execution beginnings. . . . 20
3 Indicator functions of critical sections. . . . 24
4 Sub-instances due to critical sections. . . . 27
5 Two instances needing the same two resources. . . . 31
6 Deadlock on non-preemptable resources. . . . 31
7 Round Robin: illustration of the priority components. . . . 37
8 A general MWAF . . . 46

List of Tables

1 EDF and layers. . . . 56
2 A task set under FPP only 57
3 ... and with a Round Robin layer. . . . 58
4 Non-preemptive policies in preemptive layers. . . . 58
5 Effects of critical sections under the priority ceiling protocol . . . 59
6 Same task set but without critical sections . . . 60

1 Introduction

The main property of a real-time system is feasibility. It is the guarantee that task always meet their deadlines, when scheduled according to the chosen policy. The earliest deadline first policy (EDF) is known to be optimal, for obtaining a feasible schedule. If a set of tasks is not feasible under EDF, then it is not feasible under any other scheduling policy. Feasibility may however not be the only desired property. For example, low (average) response time jitter may improve the quality of the system. If the task set is feasible under several scheduling policies, then a more refined choice should be possible. But since feasibility is the basic property, other policies can only be chosen if a corresponding feasibility test or more generally response time bounds are known. The purpose of this study is to enlarge the spectrum of analyzable scheduling policies. On one hand we derive response time bounds for the Round Robin scheduling policy. On the other hand, we propose layered priorities as a mean for combining known policies. Under POSIX 1003.1b for example, a combination through preemptive layers of fixed preemptive priorities (FPP), first in first out (FIFO) and Round Robin are allowed. We show how to derive response time bounds under layered priorities in order to enable their use in real-time systems.

An important issue besides feasibility is the appropriate handling of critical sections. A task is said to execute a critical section when it is using a non-preemptable resource. During such a period no other instance should start to use the same resource. Usually, *semaphores* are used to control the access to non-preemptable resources, but without an additional mechanism such as the *priority ceiling protocol*, nested critical section can result in deadlocks. For this reason, we will present the priority ceiling protocol in a generic in terms of priority functions. Under this form it can be applied to any preemptive policy, in order to guarantee that no deadlocks occur. We apply it to Round Robin and layered priorities. Furthermore, we show the link between critical sections and non-preemptable resources and use it to define non-preemptive Round Robin.

To define layered priorities formally and to derive response time bounds, we will use the trajectory based model and the concept of priority function introduced in [8]. We will see that Round Robin and non-preemptive versions of preemptive policy can be realized through *time dependent* priority functions, in opposition to FPP and EDF which are based on *time independent* priority functions [8]. Furthermore, we extend the model by functions that express the resource requirements of tasks. We use it in particular for non-preemptive resources and the associated priority ceiling protocol.

2 Framework

This section describes the mathematical model which has been introduced in [8] to represent a processor that executes recurrent task under real-time constraints. For further details, and the omitted proofs, the reader is referred to [8].

2.1 The task process

In this section we define the concept of *task process*. Its purpose is to describe the possible evolution of the demands of tasks that are to be scheduled. We also introduce the concept of work arrival function to formalize these demands. The scheduling itself is represented by the concept of *scheduled task process*, defined in the subsequent Section 2.2.

2.1.1 Recurrent tasks

The system to be modeled consists in a finite set of *recurrent tasks* $\mathcal{T} = \{\tau_1, \dots, \tau_m\}$ and a processing unit that executes the successive *instances* or *invocations* of these tasks. We denote by $\tau_{k,n}$ the n^{th} instance of task τ_k and by $A_{k,n} \in \mathbb{R}_+$, with $n \in \mathbb{N}$, the corresponding *activation* or *release* time. For each task τ_k, the arrivals are assumed to be indexed such that:

$$0 \leqslant A_{k,0} \leqslant A_{k,1} \leqslant \ldots \leqslant A_{k,n-1} \leqslant A_{k,n} \leqslant A_{k,n+1} \leqslant \ldots \tag{2.1}$$

i.e. in the order of their occurrence, with some given order among synchronous arrivals. The time between two successive arrivals, the *inter arrival time* or *cycle time* is denoted $T_{k,n} \stackrel{\mathsf{def}}{=} A_{k,n+1} - A_{k,n}$. We assume that the sequence of invocations of a task has no *point of accumulation*, i.e. for each $\tau_k \in \mathcal{T}$ there are a finite number of activations in any interval of finite length.

$$\forall\, t_1 \leqslant t_2 \in \mathbb{R}_+ \;: \qquad \sum_{n \in \mathbb{N}} \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]} < \infty. \tag{2.2}$$

Since each task has by definition infinitely many instances, (2.2) implies that tasks really are recurrent in the sense that for any time $t \in \mathbb{R}_+$ there exists a release of each task after t.

Let us denote $\mathbb{A} \subset (\mathbb{R}_+^{\mathbb{N}})^m$ the m-dimensional set of sequences satisfying (2.1) and (2.2). With a release $A_{k,n}$ is associated its *execution time* $C_{k,n} \in \mathbb{R}_+$, the amount of work to be done to complete the instance and its *absolute deadline* $D_{k,n}$, the time before which the task should complete its execution. It is related to the activation time by the *relative deadline* denoted $\overline{D}_{k,n}$:

$$D_{k,n} \stackrel{\mathsf{def}}{=} A_{k,n} + \overline{D}_{k,n}.$$

It is reasonable to assume that $\overline{D}_{k,n} \geqslant 0$ because otherwise, a task may be activated after its deadline.

We call $\mathbb{T} \stackrel{\mathsf{def}}{=} \mathbb{A} \times (\mathbb{R}_+^{\mathbb{N}})^m \times (\mathbb{R}_+^{\mathbb{N}})^m$ the *set of task sequences.* The elements of $\mathbb{T}$ represent all possible ways in which instances of m recurrent tasks can be activated and request the processing unit in the time interval $[0, \infty[$.

The behavior of a real-world task set is modeled by a subset of $\mathbb{T}$, i.e. a certain set of task sequences that represent all possible behaviors of the tasks. To be able to identify in $\mathbb{T}$ the task sequences that correspond to the modeled system, we introduce an index-set Ω, which might be countable or not. We use a mapping

$$(A, C, D) :\ \Omega \longrightarrow \mathbb{T} :\ \omega \longmapsto (a, c, d) \tag{2.3}$$

to pick out the concerned task sequences from $\mathbb{T}$. Notice that we denote the function by (A, C, D), to remind that it is composed of three components "activation sequences", "execution time sequences" and "deadline sequences". An element $\omega \in \Omega$ is called *trajectory* or *history* of the system. In a similar way as a random variable represents the different possible outcomes of a random experiment, (A, C, D) represents the different possible evolutions that correspond to the modeled system.

Definition 2.1 *A mapping (A, C, D) from a set Ω into the set of task sequences $\mathbb{T}$ is called a* task process.

Notice that in the terminology of *point processes*, activation times are the *points* and execution times and deadlines are *marks.*

We should for example write $A_{k,n}(\omega)$ for the n^{th} release of task τ_k on trajectory ω, but in order to avoid cumbersome notations, we will simply write $A_{k,n}$, unless making appear ω explicitly is needed.

As with random variables we write in small letters a particular realization of the task process. A task sequence in $\mathbb{T}$, that is an arrival time sequence together with an execution time sequence will be denoted (a, c, d). In order to keep notations sparse, we will avoid as far as possible to address particular realizations. Definitions and properties are stated in terms of task processes - written with capital letters - under the convention that they are meant for all trajectories. But trajectories are explicitly written where this convention leads to ambiguities or if this is required for any other reason.

2.1.2 Work arrival functions (WAF)

The response time of an instance of a task depends on the concurrent demands of other instances, belonging to the same task or not. Demands become effective after the corresponding release time $A_{k,n}$, meaning that an instance can not be executed before that time. To represent the amount of requested execution time of an instance $\tau_{k,n}$, if there is an effective demand, we introduce

$$S_{k,n}(t_1,t_2) \stackrel{\mathsf{def}}{=} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]} \tag{2.4}$$

for any interval $[t_1, t_2[$. We call $S_{k,n}$ the *work arrival function* (WAF for short) of an instance. It measures the amount of work due to instance $\tau_{k,n}$ and arriving during the interval $[t_1, t_2[$. It is a very simple function, that is of minor interest as such, but it serves as building brick for other functions. Let $\mathcal{I}$ be a set of instances (not necessarily belonging to the same task). The corresponding WAF is defined by

$$S_{\mathcal{I}}(t_1,t_2) \stackrel{\mathsf{def}}{=} \sum_{\tau_{i,j} \in \mathcal{I}} S_{i,j}(t_1,t_2). \tag{2.5}$$

An example is the WAF of a given task τ_k, which corresponds to $\mathcal{I} = \{\tau_{k,n} \mid n \in \mathbb{N}\}$:

$$S_k(t_1,t_2) \stackrel{\mathsf{def}}{=} \sum_{n \in \mathbb{N}} S_{k,n}(t_1,t_2) = \sum_{n \in \mathbb{N}} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]}. \tag{2.6}$$

Notice that this function is piecewise constant, meaning that any interval of finite length can be subdivided into a finite number of subintervals on which the function is constant. This is partially due to the fact that the sequence of activation times is assumed to have no points of accumulation, see (2.2). We will call a piecewise constant function also a *step-function*. A WAF is in particular an increasing step-function in its second variable. Notice that these functions are left-continuous in their second variable which means that an arrival at t_2 is not taken into account by $S_{k,n}(t_1, t_2)$. In order to have possible releases at t_2 taken into account, we do not need to introduce a new function. We can use the right-hand side limits in the second variable. To see this, notice first that $[t_1, t_2] = \bigcap_{\varepsilon > 0} [t_1, t_2 + \varepsilon[$. Thus,

$$C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} \leqslant t_2]} = \lim_{\varepsilon \to 0+} S_{k,n}(t_1, t_2 + \varepsilon) = S_{k,n}(t_1, t_2^+). \tag{2.7}$$

In a similar way, the right-hand side limits in the first variable implies that possible arrivals at t_1 are not taken into account: $S_{k,n}(t_1^+, t_2)$. Which of the variants is needed, depends on the context. Notice that on contiguous intervals, for example $[t_1, t_2[\cup [t_2, t_3[$, WAF's are additive:

$$S_k(t_1,t_2) + S_k(t_2,t_3) = S_k(t_1,t_3) \qquad \forall\, t_1 \leqslant t_2 \leqslant t_3. \tag{2.8}$$

In connection with feasibility it is useful to consider sets of instances $\mathcal{I}$ that are determined by a deadline d. The *deadline based work arrival function* a given task τ_k, which corresponds to $\mathcal{I} = \{\tau_{k,n} \mid n \in \mathbb{N},\ D_{k,n} \leqslant d\}$ is defined by:

$$S_k(t_1,t_2,d) \stackrel{\mathsf{def}}{=} \sum_{n \in \mathbb{N}} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]} \cdot \mathbb{1}_{[D_{k,n} \leqslant d]}. \tag{2.9}$$

For the analysis of other scheduling policies than those considered in this document, it might be necessary to introduce further kinds of WAF, based on other sets of instances.

2.2 The scheduled task process

The processing capacity or speed of the processing unit is shared by the instances according to a given scheduling policy. In this section we introduce the *scheduling function*, which is an exact representation of the way in which the policy attributes the processing unit to tasks. We also define the resulting responses times of instances of a task.

2.2.1 The scheduling function

Since we only consider one processing unit at the same time, we can choose a unit capacity and assume that execution times and other characteristics are expressed in the corresponding time unit, to simplify the notations. The effect of the policy is "materialized" by the *scheduling function* Π that gives for every instance $\tau_{k,n}$ the part

$$\Pi_{k,n}(t) \in [0,1]$$

of the processor capacity that the scheduler attributes to $\tau_{k,n}$.

Several assumptions have to be made about Π. Each function $\Pi_{k,n}$ is assumed to be *right-continuous* in t which implies that the $\Pi_{k,n}$ are Lebesgue-measurable. The following assumptions have interpretations in terms of the system to be modeled. The first is:

$$\forall\, t < A_{k,n} \qquad \Pi_{k,n}(t) = 0, \tag{2.10}$$

which means that an invocation can not use the processor before its activation date. Furthermore

$$\int_{t_1}^{t_2} \Pi_{k,n}(t)dt \leqslant C_{k,n} \qquad \forall\, t_1, t_2 \in \mathbb{R}, \tag{2.11}$$

which means that an invocation can not use the processor for longer than its nominal execution time, given unit processor speed. And finally

$$\forall\, t \in \mathbb{R}_+ \quad \textstyle\sum_{k=1}^{m} \sum_{n\in\mathbb{N}} \Pi_{k,n}(t) \leqslant 1, \tag{2.12}$$

meaning that the tasks can not use more capacity than available at time t. Let us denote $\mathcal{F}$ the set of functions Π satisfying (2.12). To represent the effect of a scheduling policy, we introduce the set $\mathbb{T}^{\Pi} \subset \mathbb{T} \times \mathcal{F}$ of *scheduled tasks sequences*, satisfying (2.10) and (2.11).

Definition 2.2 *A mapping (A, C, D, Π) from Ω into $\mathbb{T}^{\Pi}$ is called* scheduled task process.

A scheduled task process can represent *random* scheduling policies which take their decisions "by tossing a dice", but we will not consider such policies here. We are only interested in policies that apply a fixed rule. These policies are *deterministic* in the sense that the same task sequence (a, c, d) can be scheduled in only one way, i.e. is transformed into exactly one scheduled task sequence (a, c, d, π) and hence the abstract concept "deterministic policy" can be represented as a function from $\mathbb{T}$ into $\mathbb{T}^{\Pi}$.

Definition 2.3 *A mapping ν from a $\mathbb{T}$ into $\mathbb{T}^{\Pi}$ is called a* deterministic scheduling policy.

With this definition, a task process is said to be scheduled according to policy ν if

$$(A, C, D, \Pi) = \nu((A, C, D)). \tag{2.13}$$

The scheduling function Π of a task process does depend on ω because the task process is a function of ω, but not because of the scheduling policy. For random policies it would be necessary to introduce a "stochastic driver", that is a random variable which represents the random part of the scheduling policy.

2.2.2 Response times

The basic structure being settled, we can define the quantities we are interested in, such as *response times*, *execution beginnings* and *execution ends.* ends. A very useful tool for their analysis are workload functions, defined by: *workload function*

$$W_{k,n}(t) \stackrel{\text{def}}{=} S_{k,n}(0,t) - \int_0^t \Pi_{k,n}(u)du. \tag{2.14}$$

For each instance $\tau_{k,n}$ it represents the amount of work which has arrived before t and which is still waiting for execution. Notice that $W_{k,n}(A_{k,n}) = 0$ because $S_{k,n}(0, A_{k,n})$ does not account for the arrival at $A_{k,n}$. This property corresponds to the left-continuity of $W_{k,n}(t)$, which is induced by the continuity of the integral of $\Pi_{k,n}$ and the left-continuity of $S_{k,n}(0,t)$. Taking the right-hand side limit in t gives:

$$W_{k,n}(t^+) = S_{k,n}(0,t^+) - \int_0^t \Pi_{k,n}(u)du. \tag{2.15}$$

It represents the pending work at t whether it arrives before or at t.

Definition 2.4 *An instance $\tau_{k,n}$ is said to be* pending *at a time t if $W_{k,n}(t^+) > 0$.*

Notice that a task can only be executed if it is pending:

$$\Pi_{k,n}(t) > 0 \Rightarrow W_{k,n}(t) > 0. \tag{2.16}$$

It is implied by assumptions (2.10) and (2.11). Notice also the following property:

Lemma 2.5 *The number of pending instances is finite on any interval of finite length:*

$$\forall\, t_1 < t_2 \in \mathbb{R}_+ \qquad \sum_{\tau_{k,n}} \mathbb{1}_{[W_{k,n}(t^+)>0]} \cdot \mathbb{1}_{[t_1 \leqslant t < t_2]} < \infty. \tag{2.17}$$

It is sometimes necessary to consider only instances that have deadlines earlier than some deadline d. For this purpose we define deadline based version of the involved functions:

$$W_{k,n}(t,d) \stackrel{\text{def}}{=} W_{k,n}(t) \cdot \mathbb{1}_{[D_{k,n} \leqslant d]} \quad S_{k,n}(0,t,d) \stackrel{\text{def}}{=} S_{k,n}(0,t) \cdot \mathbb{1}_{[D_{k,n} \leqslant d]} \quad \Pi_{k,n}(t,d) \stackrel{\text{def}}{=} \Pi_{k,n}(t) \cdot \mathbb{1}_{[D_{k,n} \leqslant d]}.$$

The sum over all instances of a task gives $W_k(t,d)$, $S_k(0,t,d)$ and $\Pi_k(t,d)$. It can easily be seen that

$$W_{k,n}(t,d) = S_{k,n}(0,t,d) - \int_0^t \Pi_{k,n}(u,d)du.$$

A pending instance completes its execution as soon as its workload vanishes. Thus, the *execution end* or *completion time* of an instance can be defined by

$$E_{k,n} \stackrel{\text{def}}{=} \min\{t > A_{k,n} \mid W_{k,n}(t) = 0\}, \tag{2.18}$$

if it exists, that is if the set on which the definition is based on is not empty. Notice that between $A_{k,n}$ and $E_{k,n}$, the execution of $\tau_{k,n}$ could be interrupted if the scheduling policy is preemptive and thus, the *response time*, defined by

$$R_{k,n} \stackrel{\text{def}}{=} E_{k,n} - A_{k,n} \tag{2.19}$$

can be longer than the execution time $C_{k,n}$.

The *execution begin* of an instance can be defined as the first time after its activation, where its scheduling function becomes positive:

$$B_{k,n} \stackrel{\text{def}}{=} \inf\{t \geqslant A_{k,n} \mid \Pi_{k,n}(t) > 0\}. \tag{2.20}$$

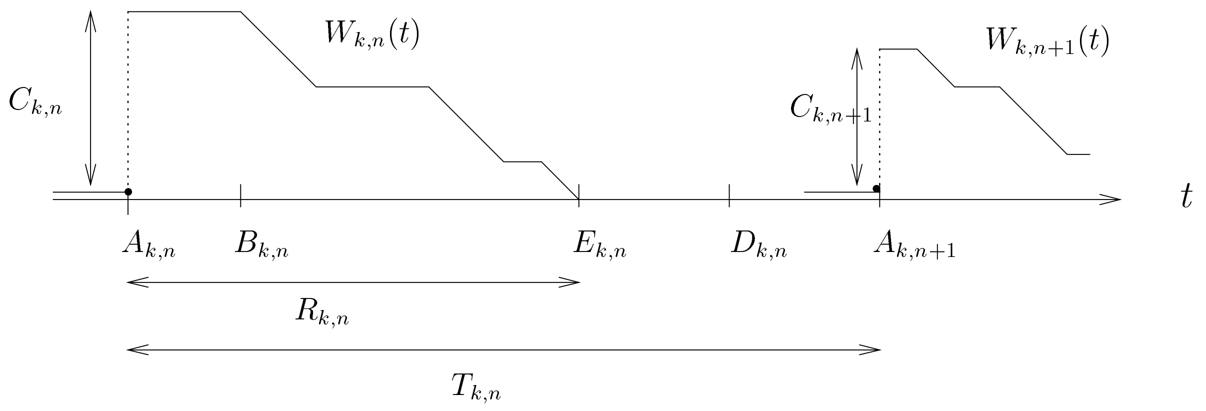


Figure 1: Times and functions associated with the instance of a task τ_k.

2.2.3 Feasibility

Under real-time constraints, tasks must complete their execution within their *relative deadline* $\overline{D}_{k,n}$, i.e. before their *absolute deadline* $D_{k,n} = A_{k,n} + \overline{D}_{k,n}$. If all instances meet their deadlines then the task set is *feasible*. In our model feasibility is defined a follows:

Definition 2.6 *A set of tasks is* feasible *under a certain scheduling policy, if in the corresponding scheduled task process* (A, C, D, Π), *all task instances finish before their deadline:*

$$E_{k,n}(\omega) \leqslant D_{k,n}(\omega) \qquad \forall \omega \in \Omega, \forall k = 1..m, \forall n \in \mathbb{N}.$$

Notice that feasibility only concerns the property that response times are shorter than relative deadlines. On the other hand, maximal response times or bounds on response times, provide additional information. They tell how close to the deadline response times could come. Close could be below or above. If a task is not feasible, it allows to know by how much a deadline is not met and if it is feasible it gives an idea about the safety margin. These are useful informations for the design of a system. If the relative deadlines of the instances of a task are all equal, $D_{k,n} = \overline{D}_k \ \forall n \in \mathbb{N}$, then the maximal response times of a task allows to say if the task is feasible or not. A task is then feasible if and only if

$$R_k^{max} \stackrel{\mathsf{def}}{=} \max_{\omega \in \Omega, n \in \mathbb{N}} R_{k,n}(\omega) \leqslant \overline{D}_k.$$

2.3 Basic assumptions

As said above, execution ends or even beginnings might be undefined. This can happen if the schedule does not allow an instance of a task to execute. In the context of real-time system where feasibility is the seeked property, such behaviors are of course undesired. If the used scheduling is *non-idling*, meaning that the processing unit does not idle if work is pending, then the existence or non-existence of execution ends or beginnings is related to the long term load on the processing unit by task's demands. Since we are only interested in *non-idling* scheduling policies, we will impose a *deterministic stability* condition, that guarantees together with the non-idling assumption that all execution ends are defined. In this section we express these properties in the model of scheduled task processes and show which kind of basic properties they induce.

2.3.1 Deterministic stability

The aim of introducing this stability condition is to ensure that response times are finite. It does not imply feasibility but it is a necessary condition. If response times are not finite, then there is no use in trying to derive response time bounds. Furthermore, the commonly used iterative algorithm to compute response time bounds numerically is only valid if the stability condition is satisfied.

Stability can be interpreted as the guarantee that the processing unit is not overloaded by task demands in the long run. To determine if a set of tasks does or does not overload the processing unit on the long run the tasks demands must be estimated. For this purpose, *(σ,ρ)-bounds* [3] are appropriate. They are based on a slope $\rho_k \in \mathbb{R}_+$ and constant $\sigma_k \in \mathbb{R}_+$ that satisfy

$$S_k(t_1, t_2; \omega) = \sum_{n \in \mathbb{N}} S_{k,n}(t_1, t_2; \omega) \leqslant \sigma_k + \rho_k \cdot (t_2 - t_1) \qquad \forall\, t_1 \leqslant t_2 \in \mathbb{R}_+,\ \forall\, \omega \in \Omega. \tag{2.21}$$

It means that the task's WAF's are uniformly bounded by one affine function on all intervals of all trajectories. The sums $\sigma_{1..m} \stackrel{\mathsf{def}}{=} \sum_{i=1}^{m} \sigma_i$ and $\rho_{1..m} \stackrel{\mathsf{def}}{=} \sum_{i=1}^{m} \rho_i$ give a (σ,ρ)-bound for the whole task set.

Definition 2.7 *A task process is said to be* stable *if there exist (σ,ρ)-bounds such that $\rho_{1..m} < 1$. The subset of stable task sequences is denoted $\mathbb{T}_0$.*

Notice that this property is independent of the used scheduling policy. Together with the non-idling assumption this implies policy independent properties, which are presented in Section 3.1.3.

2.3.2 Non-idling

In order to state this property of scheduling policies, we introduce

$$S_{1..m}(t_1, t_2) = \sum_{i=1}^{m} S_i(t_1, t_2) \qquad W_{1..m}(t) = \sum_{i=1}^{m} W_i(t) \qquad \Pi_{1..m}(t) = \sum_{i=1}^{m} \Pi_i(t) \tag{2.22}$$

that are, respectively, the total amount of work arriving on $[t_1, t_2[$, the total pending work from the past and the total use of the processing time at time t. Notice that the considered sums potentially concern infinite numbers of terms, but since an arrival process is assumed to have no points of accumulation (2.2), $S_{1..m}$ is finite and piecewise constant in t_2, for a fixed ω, implying that its left-continuity is preserved.

Definition 2.8 *A scheduled task process is said to be* non-idling *if*

$$W_{1..m}(t) > 0 \quad \Rightarrow \quad \Pi_{1..m}(t) = 1. \tag{2.23}$$

The subset of scheduled task sequences satisfying (2.23) is denoted $\mathbb{T}_0^{\Pi}$.

It means that the processing unit is fully used if work is pending. As counter example, consider a task that has to do some pre-processing, acquisition of data (say, from a buffer) and finally the actual processing. If the task has to wait before the data is available, then an interval could occur where the processor is idling although work is pending - the work that corresponds to the actual processing. This work has to be considered as pending because the definition of the task comprises all three steps and by the definition of activation dates, the entire amount of work is considered as pending as soon as the task is activated. In such a case, the initial task would have to be subdivided into two tasks with a precedence relation.

2.3.3 Majorizing work arrival functions (MWAF)

In order to derive response time bounds and to test feasibility, it is necessary to be able to bound the demands of each task. An example are the (σ,ρ)-bounds introduced in Section 2.3.1. In a more general way, we give in this section the definition of *families of majorizing work arrival function*, a concept that allows to express these bounds appropriately in our model. We discussed in [8] the connections with the well known concept of *critical instant.*

To bound the response time of an instance or to check feasibility, it is necessary to bound the concurrent demands from other tasks. To formalize this, we use the concept of *majorizing work arrival function* [8].

Definition 2.9 *A family of majorizing work arrival functions for a set of tasks is a set of functions* $\mathcal{S} = \{\widehat{S}_k(x,q) \,|\, k = 1..m,\ q \in Q\}$ *such that for any time* u *and trajectory* ω*, there exists an index* $q \in Q$ *such that the WAF of each task is bounded by the corresponding MWAF:*

$$\forall u \in \mathbb{R}_+,\ \forall \omega \in \Omega\ \exists q \in Q \qquad S_k(u, u+x;\, \omega) \leqslant \widehat{S}_k(x,q)\ \ \forall\, x > 0,\ k = 1, \ldots, m.$$

Definition 2.10 *A family of majorizing deadline based work arrival functions for a set of tasks is a set of functions* $\mathcal{S} = \{\widehat{S}_k(x,d,q) \mid k = 1..m,\ q \in Q\}$ *such that for any time* u *and trajectory* ω*, there exists an index* $q \in Q$ *such that the WAF of each task is bounded by the corresponding MWAF:*

$$\forall u \in \mathbb{R}_+,\ \forall \omega \in \Omega\ \exists q \in Q \qquad S_k(u, u+x, u+d;\, \omega) \leqslant \widehat{S}_k(x,d,q)\ \ \forall\, x > 0, k = 1, \ldots, m.$$

2.4 Highest Priority First Scheduling

So far we have only considered the effects of scheduling policies via the scheduling function Π. In this section we do a first step in the direction of defining concrete policies by introducing the class of policies that allows only one task to use the processing unit at the same time. Such policies can be described within the *highest priority first* paradigm (HPF). A straightforward example is the fixed (or static) priority preemptive scheduling policy. There are many other policy where the processing unit can be used by only one task at the same time: earliest deadline first (EDF), first in first out (FIFO), Round Robin (RR), etc. To allow all these policies to be described within HPF, we introduce a concept *priority assignment*, which gives to each instance of a task a multidimensional priority. This priority assignment enables us to write general properties, which are valid for every scheduling policy of the class.

2.4.1 Motivation and definition

Consider a processing unit that can be used by only one instance at the same time, i.e. such that

$$\Pi_{k,n}(t) \in \{0,1\} \qquad \text{and} \qquad \sum_{k,n} \Pi_{k,n}(t) \in \{0,1\}. \tag{2.24}$$

In this case, the scheduling policy could be based on a mechanism that designates at each time exactly one instance among those which are active. For this purpose we introduce a *priority assignment* Γ, that gives the *priority* of every instance at any time t:

$$\Gamma_{k,n}(t) \in \mathcal{P}. \tag{2.25}$$

To promote flexibility, the priority space $(\mathcal{P}, \preccurlyeq)$ may be any set of totally ordered elements. For the policies studied in this document we choose as priority space the set of multidimensional $\mathbb{R}$-valued vectors $\mathcal{P} = \{(p_1, \ldots, p_n) \in \mathbb{R}^n \mid n \in \mathbb{N}\}$. Two vectors are equal if they have the same size and if the components are equal one by one:

$$(p_1, \ldots, p_n) = (p'_1, \ldots, p'_{n'}) \quad \Longleftrightarrow \quad n = n' \text{ and } p_j = p'_j,\ j = 1, \ldots, n. \tag{2.26}$$

We write $p \preccurlyeq p'$ to say that priority p is smaller than or equal to priority p' and $p \prec p'$ to say that p is strictly smaller than p'. The relation "$p \prec p'$", that is "$p \preccurlyeq p'$ and $p \neq p'$" is defined by

$$(p_1, \dots, p_n) \prec (p'_1, \dots, p'_{n'}) \quad \Longleftrightarrow \quad \left| \begin{array}{l} \exists\, i \leqslant \min(n, n') \vdash \quad p_j = p'_j,\ j < i \text{ and } p_i > p'_i \\ \text{or} \\ n' < n \text{ and } p_j = p'_j,\ j = 1, \dots, n'. \end{array} \right. \tag{2.27}$$

For example: $(1,1,4,9,2) \prec (1,1,3,5,6)$ and $(1,1,4,9,2) \prec (1,1,4)$, but $(1,1,3,9,2) \succ (1,1,4)$.

We allow vectors with different lengths, because it can not be foreseen how many coordinates will actually be needed in a specific context. Notice that (2.27) defines a total order $\mathcal{P}$.

A priority assignment must be *decidable* that is such that at any time there is exactly one instance with maximal priority among the currently active instances. This implies that they have to be different:

Definition 2.11 *A priority assignment is said to be decidable, if*

$$t \in \mathbb{R}_+,\ \tau_{k,n} \neq \tau_{k',n'} \text{ such that } W_{k,n}(t^+) > 0 \text{ and } W_{k',n'}(t^+) > 0 \quad \Rightarrow \quad \Gamma_{k,n}(t) \neq \Gamma_{k',n'}(t). \tag{2.28}$$

Definition 2.12 *A non-idling scheduled task process is said to satisfy the* highest priority first *rule (HPF, for short) for a priority assignment Γ if for all $\tau_{k',n'} \neq \tau_{k,n}$ and $\forall\, \omega \in \Omega$*

$$\Pi_{k,n}(t) = 1 \text{ and } W_{k',n'}(t) > 0 \quad \Rightarrow \quad \Gamma_{k,n}(t) \succ \Gamma_{k',n'}(t). \tag{2.29}$$

The question that arises now, is under which assumptions about priority assignments the HPF-rule determines a scheduling policy. This question is not only motivated by theoretical considerations. The answer tells how a policy can be defined that will always work properly, meaning in this context that the processing unit neither idles, i.e. does not execute any instance although tasks are pending, nor tries to execute two instances at the same time, although it should not, see assumption (2.24). We do not know which conditions are necessary, but we know a reasonable sufficient condition:

Definition 2.13 *A priority assignment Γ is said to be* piecewise order preserving *if for any finite interval $[t_0, t_I[$ there is a finite partition $t_0 < t_1 < \dots < t_I$ such that for all $\tau_{k,n}$, $\tau_{k',n'}$, $i \in \{1, \dots, I\}$*

$$\Gamma_{k,n}(t_{i-1}) \succ \Gamma_{k',n'}(t_{i-1}) \quad \Rightarrow \quad \forall x \in [t_{i-1}, t_i[\ \ \Gamma_{k,n}(x) \succ \Gamma_{k',n'}(x). \tag{2.30}$$

Consequently, if a priority assignment does not satisfy this definition, then there exists an interval of finite length where the priority order changes infinitely often. No physical processing unit can realize such a scheduling and thus requiring the assignment to be piecewise order preserving is not restrictive for applications. Thus, although this property is probably not necessary and only sufficient it is not a loss of generality to consider only such assignments. Furthermore, the scheduling can be constructed step by step, as a real-world scheduler would. This appears in the proof of Theorem 2.15, stated below. The proof can be found in [8].

The construction is based on an identity (2.31) which induces all the properties that a scheduling function must have under HPF:

Lemma 2.14 *If the scheduling functions $\Pi_{k,n}$, and a decidable priority assignment Γ, with right-continuous $\Gamma_{k,n}$ satisfy for all $t \in [t_1, t_2[$, $k = 1..m$ and $n \in \mathbb{N}$*

$$\Pi_{k,n}(t) = \mathbb{1}_{[W_{k,n}(t^+)>0]} \cdot \prod_{\tau_{i,j} \neq \tau_{k,n}} \left(1 - \mathbb{1}_{[W_{i,j}(t^+)>0]} \cdot \mathbb{1}_{[\Gamma_{i,j}(t) \succ \Gamma_{k,n}(t)]}\right) \tag{2.31}$$

then on $[t_1, t_2[$, each $\Pi_{k,n}$ is right-continuous and satisfies the basic assumptions (2.10), (2.11) *and* (2.12)*, the exclusive use assumption* (2.24) *and the non-idling assumption* (2.23)*. Furthermore, Π and Γ satisfy the HPF-rule* (2.29).

Proof: Let $t \in [t_1, t_2[$:

- non-idling: if $W_{1..m}(t^+) > 0$, then at least one instance is pending at t. By decidability, one of the pending instances has the highest priority. Suppose it is $\tau_{k,n}$, i.e.

  $$\tau_{i,j} \neq \tau_{k,n} \text{ and } W_{i,j}(t) > 0 \quad \Rightarrow \quad \Gamma_{i,j}(t) \prec \Gamma_{k,n}(t).$$

  Then, (2.31) implies $\Pi_{k,n}(t) = 1$, since $W_{k,n}(t^+) > 0$.

- exclusive use: since (2.31) is the product of factors with values in $\{0,1\}$, also $\Pi_{k,n}(t) \in \{0,1\}$ is satisfied.

  It remains to prove that $\sum \Pi_{k,n}(t) \in \{0,1\}$. If an instance is not pending, i.e. $W_{k,n}(t^+) = 0$, then (2.31) implies $\Pi_{k,n}(t) = 0$. Among the instances that are pending, one has the highest priority because Γ is assumed to be decidable. Suppose it is $\tau_{i,j}$. For any other pending instance $\tau_{k,n}$, $\Gamma_{i,j}(t) \succ \Gamma_{k,n}(t)$ is true and thus one of the factors in (2.31) is equal to zero. Hence at most one instance can have a scheduling function equal to 1.

- (2.12): it is implied by the exclusive use property.

- (2.10): is satisfied since $W_{k,n}(t^+) = 0$, for $t < A_{k,n}$ and thus $\Pi_{k,n}(t) = 0$.

- (2.11): Because of (2.14), $\int_0^t \Pi_{k,n}(x)dx = C_{k,n}$ implies $W_{k,n}(t^+) = 0$, which induces then with (2.31) that $\Pi_{k,n}(t) = 0$.

- HPF: If $\Pi_{k,n}(t) = 1$ then (2.31) implies for all $\tau_{i,j} \neq \tau_{k,n}$, that either $W_{i,j}(t^+) = 0$ or $W_{i,j}(t^+) > 0$ and $\Gamma_{i,j}(t) \prec \Gamma_{k,n}(t)$. Thus (2.29) holds.

- right-continuity: we have to prove that for every $\epsilon > 0$ there exists a $\mu > 0$ such that

  $$x < \mu \quad \Rightarrow \quad |\Pi_{k,n}(t+x) - \Pi_{k,n}(t)| < \epsilon.$$

  First notice that by Lemma 2.5, only a finite number of instances are pending in the interval $[t_1, t_2[$ and thus except for a finite number, the factors in the product in the right-hand side of (2.31) are constant. We will show that for any $t \in [t_1, t_2[$ there exists for each non-constant term a $\mu_h > 0$ such that it is (also) constant on $[t, t + \mu_h[$. Taking the minimum $\mu = \min_h \mu_h$ of the finite number of μ_h gives an interval $[t, t + \mu[$ where $\Pi_{k,n}$ is constant, which implies right-continuity. The factors in (2.31) depend on $W_{i,j}(t^+)$, $\Gamma_{i,j}(t)$ and $\Gamma_{k,n}(t)$.

  1. By definition, $W_{i,j}(t^+)$ is right-continuous. Thus, if $W_{i,j}(t^+) = 0$ then $W_{i,j}(x^+) = 0$ for $x \in [t, t + \mu_h[$ and some $\mu_h > 0$. If $W_{i,j}(t^+) > 0$ then $W_{i,j}(x^+) > 0$ for $x \in [t, t + \mu_h[$ and some $\mu_h > 0$. On this interval $W_{i,j}$ will not change the value of $\Pi_{k,n}$.
  2. By decidability, for two pending instance τ_k, τ_i, either $\Gamma_{i,j}(t) \prec \Gamma_{k,n}(t)$ or $\Gamma_{i,j}(t) \succ \Gamma_{k,n}(t)$. Because priority functions are supposed to be right-continuous, there exists a $\mu_h > 0$ such that on $[t, t+\mu_h[$ the order remains the same, and thus the priority functions do not provoke a change of $\Pi_{k,n}$.

■

Below we give the theorem used in [8] to prove that time independent priority assignments do indeed define scheduling policies. For the policies considered in this document, the Theorem must be adapted, see Section 3.1.1 for priorities with promotion at execution beginning (PPEB) and Section 3.3 for Round Robin (RR).

Theorem 2.15
For any decidable and piecewise priority order preserving assignment Γ there exists exactly one non-idling scheduling function Π that satisfies the HPF-rule, i.e Γ specifies a non-idling scheduling policy. The scheduling function and the priority assignment satisfy (2.31).

Examples of policies defined by time independent priority assignments, i.e. that satisfy

$$\Gamma_{k,n}(t) = \Gamma(0) \qquad \forall\, t \tag{2.32}$$

are

fixed preemptive priorities (FPP)	$\Gamma_{k,n}(t) = (k, n)$
earliest deadline first (EDF)	$\Gamma_{k,n}(t) = (D_{k,n}, k, n)$
first in first out (FIFO)	$\Gamma_{k,n}(t) = (A_{k,n}, k, n)$
last in first out (LIFO)	$\Gamma_{k,n}(t) = (-A_{k,n}, k, -n)$.

For further explanations the reader is referred to [8].

Let us consider a more sophisticated example to illustrate the required properties. Suppose we would choose

$$\Gamma_{k,n}(t) = \textstyle\int_0^t \Pi_{k,n}(x)dx.$$

With this assignment, the priority of an instance $\tau_{k,n}$ changes only while it is executed. If $\tau_{k,n}$ is executed, then $\int_0^t \Pi_{k,n}(x)dx$ increases, implying the priority decreases (recall (2.27)). This means that a scheduling policy based on this assignment would try to share the processor between the pending instances by reducing the priority of the executing instance, in order to allow other instances to become in their turn the highest priority instance.

But this assignment is not decidable since for example two instances activated at the same time $t = A_{k,n} = A_{i,j}$, would have equal priorities: $\Gamma_{k,n}(t) = \Gamma_{i,j}(t) = 0$. To achieve decidability, one can add two further coordinates, as for EDF in [8]:

$$\Gamma_{k,n}(t) = \left(\textstyle\int_0^t \Pi_{k,n}(x)dx, k, n\right).$$

However, a problem remains. Although the priorities are different for $\tau_{k,n}$ and $\tau_{i,j}$ at $t = A_{k,n} = A_{i,j}$, it is impossible to schedule the instances under HPF. Indeed, if $k < j$, then by (2.31), τ_k should be executed: $\Pi_{k,n}(t) = 1$ and $\Pi_{i,j}(t) = 0$. Recall that $\Pi_{k,n}$ is required to be right-continuous and thus for some $\epsilon > 0$ we must choose $\Pi_{k,n}(x) = 1$ for $x \in [t, t+\epsilon[$. But for any $\epsilon > 0$, it would imply

$$\Gamma_{k,n}(t+\epsilon) = (\epsilon, k, n) \ \prec\ (0, k, n) = \Gamma_{i,j}(t+\epsilon),$$

i.e. immediately after t, $\tau_{i,j}$ would have a higher priority than $\tau_{k,n}$.

Notice that the problem is actually not the lack of right-continuity, but the fact that scheduling $\tau_{k,n}$ for some time, no matter how short, makes $\tau_{i,j}$ immediately become the higher priority instance. A solution would be to allow tasks to share the processor instantaneously by choosing $\Pi_{k,n}(t) = \Pi_{i,j}(t) = 0.5$, but such a policy does not respect the exclusive use assumption.

A solution, within the currently considered class of policies, is to allow $\tau_{k,n}$ to remain the highest priority task during some minimal time after t. This can be achieved for example by

$$\Gamma_{k,n}(t) = \left(\left\lfloor \frac{\int_0^t \Pi_{k,n}(x)dx}{\Psi_{k,n}} \right\rfloor, k, n\right).$$

The parameter $\Psi_{k,n}$ and the integer part induce that the first priority coordinate remains constant while $\int_0^t \Pi_{k,n}(x)dx \in [h \cdot \Psi_{k,n}, (h+1) \cdot \Psi_{k,n}[$, for some $h \in \mathbb{N}$.

A similar priority assignment will be used to define the *Round Robin* (RR) scheduling policy. The difference is that under RR, the processing unit is shared at task level and not at the level of instance as in this example.

In the examples considered so far, priority functions only depend on the past. One could consider an EDF policy where the deadline of an instance is equal to the activation time of the following instance:

$$\Gamma_{k,n}(t) = (A_{k,n+1}, k, n). \tag{2.33}$$

This is different from the usual EDF policy if the tasks are not strictly periodic. As can be seen, $\gamma_{k,n}(t)$ depends on the future after t. However, this policy would only make sense for a real-world system if at every release time of an instance the release time of the following instance is known.

2.4.2 Some formulas

Consider the set of instances $\mathcal{H}_{k,n}(t)$ having a higher priority than $\tau_{k,n}$ at some time $t \in [A_{k,n}, E_{k,n}[$, i.e. a time where $\tau_{k,n}$ is pending:

$$\mathcal{H}_{k,n}(t) \stackrel{\text{def}}{=} \{\tau_{i,j} \mid \Gamma_{i,j}(t) \succ \Gamma_{k,n}(t)\}. \tag{2.34}$$

Let the corresponding set of instances with lower priority be

$$\mathcal{L}_{k,n}(t) \stackrel{\text{def}}{=} \{\tau_{i,j} \mid \Gamma_{i,j}(t) \prec \Gamma_{k,n}(t)\}. \tag{2.35}$$

The only instance being in none of the two is the considered instance itself, because of the decidability assumption about the priority assignment. Notice that these sets only take the priority structure into account, independently of activation times. Such a set can contain instances that have already been executed before t or that will be activated far after t.

For a decidable priority assignment, the definition of these sets immediately implies the following two properties, for $\tau_{k,n} \neq \tau_{i,j}$:

$$\tau_{i,j} \in \mathcal{L}_{k,n}(t) \quad \Leftrightarrow \quad \tau_{k,n} \in \mathcal{H}_{i,j}(t) \tag{2.36}$$

Also, if

$$W_{\mathcal{H}_{k,n}(t)}(t) > 0 \qquad \Rightarrow \qquad \Pi_{\mathcal{H}_{k,n}(t)}(t) = 1, \tag{2.37}$$

meaning that any instance outside of a higher priority set is preempted while instances of the set are pending.

The set $\mathcal{H}_{k,n}(t)$ allows to write $B_{k,n}$ and $E_{k,n}$ in a way that shows the reason for which, most of the time, $E_{k,n} > A_{k,n} + C_{k,n}$: the preemption from higher priority instances. The defining equation (2.18) of $E_{k,n}$ is equivalent to

$$E_{k,n} = \min\{t > A_{k,n} \mid W_{k,n}(t) + W_{\mathcal{H}_{k,n}(t)}(t) = 0\}, \tag{2.38}$$

by Proposition A.5. Indeed, for $t \in [A_{k,n}, E_{k,n}[$, $W_{k,n}(t) + W_{\mathcal{H}_{k,n}(t)}(t) > W_{k,n}(t) > 0$, i.e. (A.9) holds, with $x_0 = \hat{x}_0 = A_{k,n}$. Furthermore

$$W_{\mathcal{H}_{k,n}(E_{k,n})}(E_{k,n}) = 0, \tag{2.39}$$

because otherwise by the left-continuity of workload functions, there would exist $\epsilon > 0$ such that $W_{\mathcal{H}_{k,n}(x)}(x) > 0$ for $x \in]t_0 - \epsilon, t_0]$. And then, $\Pi_{k,n}(x) = 0$ which would imply $W_{k,n}(t_0 - \epsilon) = W_{k,n}(t_0) = 0$. Thus, (A.10) is satisfied.

The execution end can also be expressed in terms of scheduling functions:

$$E_{k,n} = \min\{t > A_{k,n} \mid C_{k,n} + \int_{A_{k,n}}^{t} \Pi_{\mathcal{H}_{k,n}(x)}(x)dx + A_{k,n} = t\}. \tag{2.40}$$

To prove this, reconsider, the defining equation (2.18) on page 9. For $t > A_{k,n}$ we have

$$W_{k,n}(t) = C_{k,n} - \int_{A_{k,n}}^{t} \Pi_{k,n}(x)dx.$$

For all $t \in [A_{k,n}, E_{k,n}[$, $W_{k,n}(t) > 0$, implying $W_{1..m}(t) > 0$ and thus $\Pi_{1..m}(t) = 1$. On the other hand, $W_{k,n}(t) > 0$ implies $\Pi_{\mathcal{L}_{k,n}(t)}(t) = 0$. Thus $\Pi_{k,n}(t) = 1 - \Pi_{\mathcal{H}_{k,n}(t)}(t)$.

The execution begin formula can also be expressed using $\mathcal{H}_{k,n}(t)$. From (2.20) on page 9 and (2.31) on page 13 it follows that

$$B_{k,n} = \min\{t \geqslant A_{k,n} \mid W_{\mathcal{H}_{k,n}(t)}(t^+) = 0\}. \tag{2.41}$$

It means that the execution begin is the first time after $A_{k,n}$ where the workload of pending instances having currently a higher priority becomes zero.

These equations will serve as starting points for deriving response time bounds under different scheduling policies.

2.4.3 Properties of priority assignments

A priority assignment which realizes FPP is for example $\Gamma_{k,n}(t) = (k, n)$. If we would choose $\Gamma'_{k,n}(t) = k+1, n+13$ instead, the resulting schedule would be the same. More generally, different assignments can be equivalent in the sense that they produce the same schedule.

Definition 2.16 *Two priority assignments Γ and Γ, with values in priority spaces $(\mathcal{P}_1, \overset{1}{\succ})$ and $(\mathcal{P}_2, \overset{2}{\succ})$ respectively, are said to be* equivalent *if any pair of task $\tau_{k,n} \neq \tau_{i,j}$ have at any time t the same priority order:*

$$\Gamma_{k,n}(t) \overset{1}{\succ} \Gamma_{i,j}(t) \Leftrightarrow \Gamma'_{k,n}(t) \overset{2}{\succ} \Gamma'_{i,j}(t). \tag{2.42}$$

Proposition 2.17 *For a given task process, two equivalent priority assignments produce the same scheduled task process.*

Proof: By contradiction, using Lemma 2.15 and (2.31). ■

The fact that two different assignments can implement the same policy is actually a very useful fact for the construction and analysis of complex policies.

It is sometimes useful to be able to derive a response time bound by using a lower bound on the priority function of an instance. The following Proposition states that this possible.

Proposition 2.18 *Let (A, C, D, Π) be a task process, scheduled HPF according to a priority assignment Γ. Let Π' be the scheduling function corresponding to a second assignment Γ', which is the same, except for one instance $\tau_{k,n}$, for which the priority function is smaller at any time:*

$$\Gamma_{k,n}(t) \succ \Gamma'_{k,n}(t) \quad \forall t. \tag{2.43}$$

Then, the response time of $\tau_{k,n}$ is larger in the second case:

$$R_{k,n} \leqslant R'_{k,n}.$$

Proof: Relation (2.43) implies that at any moment t, the set of instances with higher priorities contains more elements under Γ':

$$\mathcal{H}_{k,n}(t) \subset \mathcal{H}'_{k,n}(t) \quad \forall t.$$

First, we prove that the scheduling function associated with $\mathcal{H}_{k,n}(t)$ is the same under both assignments:

$$\Pi_{\mathcal{H}_{k,n}(t)}(t) = \Pi'_{\mathcal{H}_{k,n}(t)}(t) \quad \forall t.$$

By contradiction: suppose the equality where wrong. Since Π and Π' are right continuous, there exists then a first time t_0 where $\Pi_{\mathcal{H}_{k,n}(t_0)}(t_0) \neq \Pi'_{\mathcal{H}_{k,n}(t_0)}(t_0)$. Notice that $W_{\mathcal{H}_{k,n}(t)}(t) = W'_{\mathcal{H}_{k,n}(t)}(t)$ for all $t \leqslant t_0$. Two cases arise:

(i) $\Pi_{\mathcal{H}_{k,n}(t_0)}(t_0) = 1$ and $\Pi'_{\mathcal{H}_{k,n}(t_0)}(t_0) = 0$: then, since $W_{\mathcal{H}_{k,n}(t_0)}(t_0) > 0$ also $W'_{\mathcal{H}_{k,n}(t_0)}(t_0) > 0$. But then $\Pi'_{\mathcal{H}_{k,n}(t_0)}(t_0) = 0$ is a contradiction with the non-idling assumption (2.23) on page 11.

(ii) $\Pi_{\mathcal{H}_{k,n}(t_0)}(t_0) = 0$ and $\Pi'_{\mathcal{H}_{k,n}(t_0)}(t_0) = 1$: then, since $W_{\mathcal{H}_{k,n}(t_0)}(t_0) = 0$ also $W'_{\mathcal{H}_{k,n}(t_0)}(t_0) = 0$. But then $\Pi'_{\mathcal{H}_{k,n}(t_0)}(t_0) = 1$, is a contradiction with the basic assumptions (2.10) or (2.11) on page 8.

Now, recall the execution end formula (2.40), which is based on scheduling functions of higher priority instances. Since scheduling functions are positive, $\Pi_{\mathcal{H}_{k,n}(t)}(t) \leqslant \Pi'_{\mathcal{H}'_{k,n}(t)}(t)$. The same relation being true for the integrals of these functions, Proposition A.4 applies when replacing $\Pi_{\mathcal{H}_{k,n}(x)}(x)$ by $\Pi'_{\mathcal{H}'_{k,n}(x)}(x)$, and thus $E_{k,n} \leqslant E'_{k,n}$. ■

3 Time dependent priority assignments

We define in this section several priority assignments and discuss their applications. On one hand we consider non-preemptive scheduling policies and the priority ceiling protocol, which is used with semaphores to protect non-preemptive resources. On the other hand, we introduce layered priorities and derive timing analysis for the Round Robin policy, which allows to analyze Posix compliant scheduling policies. As pointed out before, we have to verify that in each case the assignments do indeed define scheduling policies, recall Definition 2.3 on page 8. We prove this in Section 3.6.

3.1 Priority promotion

In this section we define priority functions with increase at execution beginning from one constant level to an other. A typical application are the non-preemptive versions of preemptive policies, treated in Section 3.4. These almost time independent priority assignments are characterized by a particular state of the system at a time of priority increase, since an instance only starts to execute, i.e. a priority changes, only if no instance with currently higher priority is pending. This is a difference with *dual priorities* [4] where the promotion dates are fixed off-line at a certain time before the deadline. It is also different from Round Robin (Section 3.3), where dates of priority change are related to the amount of time the considered task has been running. This has an impact on the way response times bounds can be derived.

The idea of priority promotion does not only apply to non-preemptive scheduling policies, but also to *critical sections*. These are sections of the code of a task where a resource is needed in an exclusive manner. Critical sections are usually protected by *semaphores* and managed according to a protocol that ensures a correct functioning. An example is the Priority Ceiling Protocol [9] (PCP). In Section 3.1.3 we show how the PCP can be obtained in our model by extending the notion of priority promotion at execution beginning.

3.1.1 Priority functions with promotion at execution beginning (PPEB)

The promotion of the priority of an instance $\tau_{k,n}$ occurs at its execution beginning $B_{k,n}$. The priority before the promotion is denoted $\vec{P}_{k,n}$ and $\vec{Q}_{k,n}$ afterwards:

$$\Gamma_{k,n}(t) = \begin{cases} \vec{P}_{k,n} & \text{if } t < B_{k,n} \\ \vec{Q}_{k,n} & \text{if } t \geqslant B_{k,n}. \end{cases} \tag{3.1}$$

This function is chosen right-continuous, because it is useful for being piece-wise order preserving, (2.30). We assume that initial priorities are different from each other

$$\vec{P}_{k,n} \neq \vec{P}_{i,j} \quad \forall \tau_{k,n} \neq \tau_{i,j} \tag{3.2}$$

and different of the priorities after promotion

$$\vec{P}_{k,n} \neq \vec{Q}_{i,j} \quad \forall \tau_{k,n} \neq \tau_{i,j} \tag{3.3}$$

unless an instance does not change its priority, i.e. $\vec{P}_{k,n} = \vec{Q}_{k,n}$ is allowed. The priorities are assumed to increase after promotion

$$\vec{P}_{k,n} \preccurlyeq \vec{Q}_{k,n}. \tag{3.4}$$

If $\vec{P}_{k,n} \succ \vec{P}_{k,n+1}$ then the order among instances of a task is FIFO.

With the priority increase at execution beginning, an instance $\tau_{i,j}$ with initialy lower priority than some other instance $\tau_{k,n}$, can get a higher priority and be executed while $\tau_{k,n}$ is pending. This is commonly called priority inversion in the literature.

Would we like to define a PPEB scheduling policy, we can not use (3.1) alone because it does not specify a unique priority function. More than one $\Gamma_{k,n}(t)$ may satisfy this equation with a corresponding scheduled task process that satisfies all required assumption. To see this, let us consider an example of two instances $\tau_{k,0}$ and $\tau_{i,0}$ with parameters:

$$\begin{array}{llll} P_{k,0} = 5 & Q_{k,0} = 2 & A_{k,0} = 1 & C_{k,0} = 2 \\ P_{i,0} = 4 & Q_{i,0} = 3 & A_{i,0} = 0 & C_{i,0} = 3. \end{array}$$

For $\Gamma_{i,0}(t)$ there is only one solution possible, because no other instance is pending when $\tau_{i,0}$ is activated and therefore necessarily $B_{i,0} = A_{i,0}$, to satisfy the non-idling assumption. As a result $\Gamma_{i,0}(t) = 3$, for $t \geqslant 0$. For $\Gamma_{k,0}(t)$ on the other hand, the following solutions are possible:

$$\Gamma_{k,0}(t) = \begin{cases} 5 & \text{if } t < 3 \\ 2 & \text{if } t \geqslant 3 \end{cases} \qquad \Gamma'_{k,0}(t) = \begin{cases} 5 & \text{if } t < 2 \\ 2 & \text{if } t \geqslant 2. \end{cases}$$

The corresponding scheduling functions are shown in Figure 2.

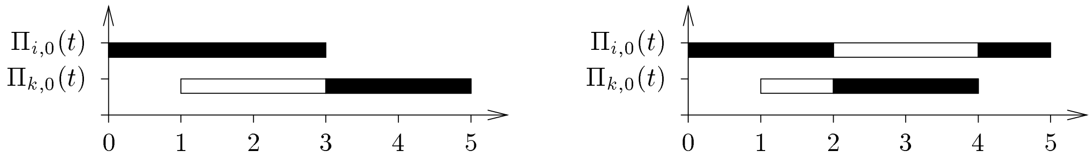


Figure 2: Two different execution beginnings.

In the first case, $B_{k,0} = E_{i,0} = 3$, which corresponds to a non-preemptive behavior, since $\tau_{k,0}$ starts to run only after $\tau_{i,0}$ has completed. In the the second case, $B'_{k,0} = 2$, because $2 = \min\{t \geqslant A_{k,0} \mid \Pi_{k,0}(t) = 1\}$, recall (2.20) on page 9. Notice that this is consistent with the HPF rule, since $\Gamma'_{k,0}(2) = 2 \succ 3 = \Gamma_{i,0}(2)$. Furthermore, $B'_{k,0} = 2$ satisfies the characteristic property

$$W_{\mathcal{H}_{k,n}(2)}(2^+) = 0$$

of execution beginnings, see (2.41) on page 17. The reason is that $\mathcal{H}_{k,n}(2) = \emptyset$, because the priority of $\tau_{k,0}$ is increased at $t = 2$. However, $W_{i,0}(2^+) > 0$.

To achieve that only the first case is a solution, we need an additional rule. We impose that

$$W_{\mathcal{H}_{k,n}(B^-_{k,n})}(B^+_{k,n}) = 0. \tag{3.5}$$

It means that none of the instances that has a priority higher than $P_{k,n}$, must be pending at the execution beginning of $\tau_{k,n}$. Since $\mathcal{H}_{k,n}(2^-) = \{\tau_{i,0}\}$, and $W_{i,0}(2^+) > 0$, $\Gamma'_{k,0}(.)$ does not satisfy (3.5) and is not a solution.

Proposition 3.1 *Equation* (3.1) *admits at most one solution that satisfies* (3.5).

Proof: Suppose there were two different solutions $\Gamma_{k,n}(.)$ and $\Gamma'_{k,n}(.)$ with $B_{k,n} < B'_{k,n}$. According to (3.5), there is no pending or just activated instance $\tau_{i,j}$ with $\Gamma_{i,j}(B_{k,n}) \succ P_{k,n}$. Since activation times have no points of accumulation, an instance with a higher priority may only be activated after $B_{k,n}$ with some strictly positive delay. Furthermore, the right-continuity of priority functions implies that another pending instance $\tau_{i,j}$ with $\Gamma_{i,j}(B_{k,n}) \prec P_{k,n}$ may only change its priority strictly after $B_{k,n}$. Thus, $B'_{k,n} > B_{k,n}$ means either a violation of the non-idling assumption of the HPF rule. ■

The existence of a solution is proven by Theorem 3.15 on page 41. Notice that because of Proposition 3.1, a PPEB priority function $\Gamma_{k,n}(.)$ is exactly specified by the two parameters $\vec{P}_{k,n}$ and $\vec{Q}_{k,n}$.

Definition 3.2 *A set of instances with priority functions of the form* (3.1) *and satisfying* (3.2), (3.3), (3.4) *and so that the scheduled task process satisfies* (3.5), *are said to have* priorities with promotion at execution beginning *(PPEB).*

Basic properties of PPEB Under PPEB, the priority order among instances changes during time. However, we will see in this section that from the point of view of a pending instance, the sets of instances with higher, resp. lower priority change only once, at its execution beginning. As a result, response times consist in two parts which have to be computed one after the other. The two parts are separated by the execution beginning:

$$B_{k,n} = \min\{t \geqslant A_{k,n} \mid W_{\mathcal{H}_{k,n}(t)}(t^+) = 0\},$$

as given by (2.41). For $t < B_{k,n}$, i.e. while $\Gamma_{k,n}(t) = \vec{P}_{k,n}$, the set $\mathcal{H}_{k,n}(t)$ contains among other instances, those having a higher initial priority than $\tau_{k,n}$:

$$\mathcal{P}_{k,n} = \{\tau_{i,j} \mid \vec{P}_{i,j} \succ \vec{P}_{k,n}\}. \tag{3.6}$$

We will see that for PPEB it is convenient to define an interference period based on $\mathcal{P}_{k,n}$:

Definition 3.3 *The time $U_{k,n}$ satisfying*

$$U_{k,n} \leqslant A_{k,n}, \quad W_{\mathcal{P}_{k,n}}(U_{k,n}) = 0 \qquad \text{and} \qquad W_{\mathcal{P}_{k,n}}(t) > 0 \quad \forall\, t \in]U_{k,n}, A_{k,n}] \tag{3.7}$$

is called beginning of the interference period $[U_{k,n}, E_{k,n}[$ *associated with the n^{th} instance of task τ_k, in the case of priority promotion at execution beginning.*

Since $\mathcal{P}_{k,n} \subset \mathcal{H}_{k,n}(t)$, this definition implies that $W_{\mathcal{H}_{k,n}(t)}(t) > 0$, for $t \in [U_{k,n}, A_{k,n}[$. Using Proposition A.3, it can therefore be proven that

$$B_{k,n} = \min\{t \geqslant U_{k,n} \mid W_{\mathcal{H}_{k,n}(t)}(t^+) = 0\}. \tag{3.8}$$

The set $\mathcal{H}_{k,n}(t)$ may contain also other instances than those in $\mathcal{P}_{k,n}$. If for $t \in [U_{k,n}, B_{k,n}[$, $\tau_{i,j} \in \mathcal{H}_{k,n}(t) \setminus \mathcal{P}_{k,n}$, then $\tau_{i,j}$ has a lower initial priority than $\tau_{k,n}$: $\vec{P}_{i,j} \prec \vec{P}_{k,n}$. It induces that $\tau_{i,j}$ must start its execution before $U_{k,n}$:

$$A_{i,j} \leqslant B_{i,j} < U_{k,n}. \tag{3.9}$$

To prove this, suppose we had $B_{i,j} \geqslant U_{k,n}$. At any $t \in [U_{k,n}, B_{k,n}[$, there is a pending instance $\tau_{i',j'}$ with $\Gamma_{i',j'}(t) \succ P_{k,n}$. Since $\vec{P}_{i,j} \prec \vec{P}_{k,n}$, $\tau_{i',j'} \in \mathcal{H}_{i,j}(t)$. Thus, $W_{\mathcal{H}_{i,j}(t)}(t) > 0$, for all $t \in [U_{k,n}, B_{k,n}[$. It implies that $W_{\mathcal{H}_{i,j}(t^-)}(t) > 0$, for all $t \in [U_{k,n}, B_{k,n}[$, i.e. $B_{i,j} \notin]U_{k,n}, B_{k,n}[$. But then, $\Gamma_{i,j}(t) = P_{i,j}$ for $t \in]U_{k,n}, B_{k,n}[$, i.e. $\tau_{i,j} \notin \mathcal{H}_{k,n}(t)$.

It implies that $\tau_{i,j}$ is part of

$$\mathcal{B}_{k,n} \stackrel{\mathsf{def}}{=} \{\tau_{i,j} \notin \mathcal{P}_{k,n} \mid \Gamma_{i,j}(U_{k,n}) \succ \vec{P}_{k,n}\}, \tag{3.10}$$

Thus, the set of higher priority instances is time independent during $[U_{k,n}, B_{k,n}[$:

$$\mathcal{H}_{k,n}(t) = \mathcal{P}_{k,n} \cup \mathcal{B}_{k,n}, \qquad t \in [U_{k,n}, B_{k,n}[\tag{3.11}$$

We turn now to the interval $[B_{k,n}, E_{k,n}[$. At the execution beginning $B_{k,n}$, the priority of $\tau_{k,n}$ is increased and thus $\mathcal{H}_{k,n}(t)$ changes. It contains those instances that always have a higher priority than $\tau_{k,n}$:

$$\mathcal{Q}_{k,n} \stackrel{\mathsf{def}}{=} \{\tau_{i,j} \mid \vec{P}_{i,j} \succ \vec{Q}_{k,n}\}. \tag{3.12}$$

If $\tau_{i,j} \in \mathcal{H}_{k,n}(t) \setminus \mathcal{Q}_{k,n}$, then $\Gamma_{i,j}(t) = \vec{Q}_{i,j} \succ \vec{Q}_{k,n}$. For a similar reason as before $\tau_{i,j}$ must have started its execution before $\tau_{k,n}$ has started to execute: $A_{i,j} < B_{k,n}$. Since $W_{\mathcal{H}_{k,n}(B_{k,n})}(B_{k,n}) = 0$, also $W_{i,j}(B_{k,n}) = 0$, i.e. $E_{i,j} \leqslant B_{k,n}$, since $\tau_{i,j}$ is not activated after $B_{k,n}$. It means that $\mathcal{H}_{k,n}(t)$ contains other instances than those in $\mathcal{Q}_{k,n}$, but their workload is zero throughout $[B_{k,n}, E_{k,n}[$. Therefore,

$$W_{\mathcal{H}_{k,n}(t)}(t) = W_{\mathcal{Q}_{k,n}}(t) \quad \text{and} \quad \Pi_{\mathcal{Q}_{k,n}}(t) + \Pi_{k,n}(t) = 1, \qquad t \in [B_{k,n}, E_{k,n}[. \tag{3.13}$$

We introduce furthermore the set of instances that move from $\mathcal{H}_{k,n}(t)$ to $\mathcal{L}_{k,n}(t)$ at $B_{k,n}$, to which we add the instance itself:

$$\mathcal{M}_{k,n} \stackrel{\mathsf{def}}{=} \{\tau_{k,n}\} \cup \mathcal{P}_{k,n} \setminus \mathcal{Q}_{k,n}. \tag{3.14}$$

Proposition 3.4 *The execution beginning of an instance with PPEB priority function, if considered relatively to the beginning of the interference period given by Definition 3.3, is:*

$$B_{k,n} = \min\{t \geqslant U_{k,n} \mid W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{P}_{k,n}}(U_{k,n}, t^+) + U_{k,n} = t\ \}. \tag{3.15}$$

The set $\mathcal{B}_{k,n}$ contains at most one pending instance at $U_{k,n}$:

$$\#\{\tau_{i,j} \in \mathcal{B}_{k,n} \mid W_{i,j}(U_{k,n}) > 0\} \leqslant 1. \tag{3.16}$$

Given the execution beginning, the execution end is

$$E_{k,n} = \min\{t > U_{k,n} \mid W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{M}_{k,n}}(U_{k,n}, B_{k,n}) + S_{\mathcal{Q}_{k,n}}(U_{k,n}, t) + U_{k,n} = t\}. \tag{3.17}$$

Proof:

- (3.15): Recall equation (3.8). Since $t \in [U_{k,n}, B_{k,n}[$, $W_{\mathcal{H}_{k,n}(t)}(t) > 0$, the basic property (2.37) on page 16 implies with (3.11) that $\Pi_{\mathcal{H}_{k,n}}(t) = \Pi_{\mathcal{P}_{k,n}}(t) + \Pi_{\mathcal{B}_{k,n}}(t) = 1$. We want to transform (3.8) into (3.15) with the help of Proposition A.5:

$$\begin{aligned} f(t) + x_0 &= W_{\mathcal{P}_{k,n}}(t) + W_{\mathcal{B}_{k,n}}(t) \\ &= W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{B}_{k,n}}(U_{k,n}, t^+) + W_{\mathcal{P}_{k,n}}(U_{k,n}) + S_{\mathcal{P}_{k,n}}(U_{k,n}, t^+) - (t - U_{k,n}) \\ &\leqslant W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{P}_{k,n}}(U_{k,n}, t^+) \stackrel{\mathsf{def}}{=} \hat{f}(t). \end{aligned}$$

  The second line is obtained using

$$W_{i,j}(t_2^+) = W_{i,j}(t_1) + S_{i,j}(t_1, t_2^+) - \textstyle\int_{t_1}^{t_2} \Pi_{i,j}(x)dx, \tag{3.18}$$

  which immediately follows from (2.14) and (2.15) on page 9. The third using:

  - by Definition 3.3, $W_{\mathcal{P}_{k,n}}(U_{k,n}) = 0$,
  - for $t_1 = U_{k,n}$, (3.9) implies $S_{\mathcal{B}_{k,n}}(U_{k,n}, t^+) = 0$ and
  - $t > U_{k,n}$.

  As a result $f(.)$ and $\hat{f}(.)$ satisfy (A.9). Since $f(B_{k,n}) = \hat{f}(B_{k,n}) = 0$, also (A.10) holds. Thus, by Proposition A.5, (3.15) is proven.

- (3.16): Assume there where two instances $\tau_{i,j} \neq \tau_{i',j'} \in \mathcal{B}_{k,n}$ with positive workload at $U_{k,n}$. As explained in (3.9), both have started before $U_{k,n}$. Since two instances can not start at the same time, let $\tau_{i,j}$ be the one starting first, i.e. $B_{i,j} < B_{i',j'} \leqslant U_{k,n}$. Then

$$\Gamma_{i,j}(B_{i',j'}) = Q_{i,j} \succ \vec{P}_{k,n} \succ \vec{P}_{i',j'}$$

  implying $\tau_{i,j} \in \mathcal{H}_{i',j'}(B_{i',j'})$. But because also $W_{i,j}(B_{i',j'}) > 0$, we have a contradiction.

- (3.17): Since $E_{k,n} > B_{k,n}$, we can rewrite (2.38) as

$$E_{k,n} = \min\{t > B_{k,n} \mid W_{k,n}(t) + W_{\mathcal{H}_{k,n}(t)}(t) = 0\}.$$

According to (3.13) and with $W_{i,j}(t_2) = W_{i,j}(t_1) + S_{i,j}(t_1, t_2) - \int_{t_1}^{t_2} \Pi_{i,j}(x)dx$ derived from (2.14), we have

$$\begin{aligned} W_{k,n}(t) + W_{\mathcal{H}_{k,n}(t)}(t) =& W_{k,n}(t) + W_{\mathcal{Q}_{k,n}}(t) \\ =& W_{k,n}(B_{k,n}) + W_{\mathcal{Q}_{k,n}}(B_{k,n}) + S_{k,n}(B_{k,n}, t) + S_{\mathcal{Q}_{k,n}}(B_{k,n}, t) \\ & - \int_{B_{k,n}}^{t} \Pi_{k,n}(x)dx - \int_{B_{k,n}}^{t} \Pi_{\mathcal{Q}_{k,n}}(x)dx. \end{aligned}$$

Thus, the execution end formula can be transformed into

$$E_{k,n} = \min\{t > B_{k,n} \mid W_{k,n}(B_{k,n}) + W_{\mathcal{Q}_{k,n}}(B_{k,n}) + S_{\mathcal{Q}_{k,n}}(B_{k,n}, t) + B_{k,n} = t\}.$$

Since $\mathcal{Q}_{k,n} \subset \mathcal{H}_{k,n}(B_{k,n})$, and $W_{\mathcal{H}_{k,n}(B_{k,n})}(B_{k,n}) = 0$, we have $W_{\mathcal{Q}_{k,n}}(B_{k,n}) = 0$. Furthermore, $W_{k,n}(B_{k,n}) = C_{k,n}$. Thus

$$E_{k,n} = \min\{t > B_{k,n} \mid C_{k,n} + S_{\mathcal{Q}_{k,n}}(B_{k,n}, t) + B_{k,n} = t\}. \tag{3.19}$$

Then, we extend the domain of t by $[U_{k,n}, B_{k,n}[$ using Proposition A.3. For $t = B_{k,n}$, (3.15) implies $B_{k,n} = W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{P}_{k,n}}(U_{k,n}, B_{k,n}^{+}) + U_{k,n}$. We notice first that actually no instance from $\mathcal{P}_{k,n}$ can be activated at $B_{k,n}$. This follows from (3.8), since $\mathcal{P}_{k,n} \subset \mathcal{H}_{k,n}$. Thus, the function $f(.)$, for the application of Proposition A.3 is

$$\begin{aligned} f(t) &= C_{k,n} + S_{\mathcal{Q}_{k,n}}(B_{k,n}, t) + B_{k,n} \\ &= W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{Q}_{k,n}}(B_{k,n}, t) + C_{k,n} + S_{\mathcal{P}_{k,n}}(U_{k,n}, B_{k,n}) + U_{k,n} \end{aligned}$$

and using (3.14) it it can be transformed to

$$\begin{aligned} &= W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{Q}_{k,n}}(B_{k,n}, t) + S_{\mathcal{M}_{k,n}}(U_{k,n}, B_{k,n}) + S_{\mathcal{Q}_{k,n}}(U_{k,n}, B_{k,n}^{+}) + U_{k,n} \\ &= W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{M}_{k,n}}(U_{k,n}, B_{k,n}) + S_{\mathcal{Q}_{k,n}}(U_{k,n}, t) + U_{k,n} \\ &\stackrel{\text{def}}{=} g(t). \end{aligned}$$

The function g is also defined for $t \in [U_{k,n}, B_{k,n}[$ where

$$g(t) = W_{\mathcal{B}_{k,n}}(U_{k,n}) + S_{\mathcal{P}_{k,n}}(U_{k,n}, t) + U_{k,n} > 0,$$

by the definition of $B_{k,n}$. Thus condition (A.4) holds and the proposition applies, which implies (3.17).

■

A time independent priority function can be seen as PPEB priority function with $\vec{P}_{k,n} = \vec{Q}_{k,n} = \Gamma_{k,n}(0)$. In the case where it is scheduled together with tasks that have PPEB priority functions, its execution time is given by a simplified version of (3.17):

Corollary 3.5 *The execution end of an instance with time independent priority function scheduled among tasks with PPEB priority functions is given by*

$$E_{k,n} = \min\{t > U_{k,n} \mid W_{\mathcal{B}_{k,n}}(U_{k,n}) + C_{k,n} + S_{\mathcal{P}_{k,n}}(U_{k,n}, t) + U_{k,n} = t\}. \tag{3.20}$$

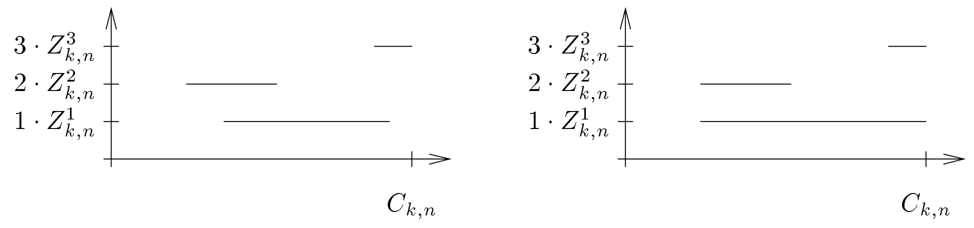
Proof: Reconsider (3.17). If $\vec{Q}_{k,n} = \vec{P}_{k,n}$ then $\mathcal{Q}_{k,n} = \mathcal{P}_{k,n}$ and thus $\mathcal{M}_{k,n} = \{\tau_{k,n}\}$. It implies that $S_{\mathcal{M}_{k,n}}(U_{k,n}, B_{k,n}^{+}) = C_{k,n}$. ■

3.1.2 Function of demand for a resource

The major resource needed by a task is the processing unit. Scheduling is in a first place concerned with sharing this resource among tasks. But tasks may also need other resources, implying additional constraints that can have an impact on the scheduling policy. In this section we introduce an indicator function that tells how a task needs a certain resource. It allows in particular to express that a resource is *non-preemptable.*

An instance may use resources during the time interval that ranges from its execution beginning until its execution end. The exact location and duration of the sub-intervals where a resource is used depends on one hand on the parts of the execution time $C_{k,n}$, (i.e. the sections of the code) that need the resource. On the other hand, it depends on the time periods where the instance is actually allowed to execute. The second cause is a result of the global scheduling policy, whereas the first only depends on the characteristics of the task. For this reason we will represent the two causes separately.

Let $Z^h_{k,n}(c)$ be a left-continuous indicator function, that tells if the resource ζ_h is required when the instance has been executed for c units of time. Figure 3 shows examples of an instance using



(a) Not properly nested (b) Properly nested

Figure 3: Indicator functions of critical sections.

three resources. To simplify the analysis it is commonly assumed that the *critical sections*, i.e. the periods where an instance needs a certain resource, are *properly nested* [9]. This property says that the corresponding resources are either not used at the same time or one is only used while the other one is used: $\forall\, c \in [0, C_{k,n}[,\ z_h,\ z_{h'}$

$$Z^h_{k,n}(c) \cdot Z^{h'}_{k,n}(c) = 0 \quad \text{or} \quad Z^h_{k,n}(c) = 1 \Rightarrow Z^{h'}_{k,n}(c) = 1 \quad \text{or} \quad Z^{h'}_{k,n}(c) = 1 \Rightarrow Z^h_{k,n}(c) = 1. \tag{3.21}$$

In Figure 3.a, critical sections are not properly nested. However they can easily be transformed by expansion such as to satisfy this assumption (Figure 3.b). Such transformations imply conservative response time bounds, and are therefore acceptable.

To be able to derive response time bounds it will be necessary to know the maximal length of a period where a task requires a resource, i.e. the maximal length of a critical section:

$$\widehat{Z}^h_k = \max\{z \mid \exists\, n,\, c_0 \ \vdash\ Z^h_{k,n}(c) = 1,\ \forall\, c \in [c_0, c_0 + z[\}. \tag{3.22}$$

Recall that $\int_0^t \Pi_{k,n}(x)dx$ is the amount of time the instance $\tau_{k,n}$ was allowed to run until t. Given the indicator functions $Z^h_{k,n}(.)$, the expression $Z^h_{k,n}(\int_0^t \Pi_{k,n}(x)dx)$ tells if at time t, $\tau_{k,n}$ *needs* ζ_h. We do not say that it uses the resource, because even while $\tau_{k,n}$ is preempted, i.e. $\Pi_{k,n}(t) = 0$, the expression remains unchanged, that is equal to one if $\tau_{k,n}$ was interrupted while using ζ_h.

Suppose that after c units of execution, an instance $\tau_{k,n}$ is about to start a critical section corresponding to ζ_h. The assumed left-continuity of $Z^h_{k,n}(.)$, implies that $Z^h_{k,n}(c) = 0$, meaning that the resource is not yet locked. However, $Z^h_{k,n}(c^+) = 1$, meaning that it is immediately locked when the instance continues to execute. Thus, the right-continuous version $Z^h_{k,n}(.^+)$ tells if $\tau_{k,n}$ needs or is about to need ζ_h.

Definition 3.6 *A left-continuous indicator function, $Z^h_{k,n}(.)$ satisfying $Z^h_{k,n}(c) = 0$ for $c \notin [0, C_{k,n}[$ is called* indicator of need *for a resource ζ_h by the instance $\tau_{k,n}$. The right-continuous version $Z^h_{k,n}(.^+)$ is called* indicator of imminent need *for resource ζ_h.*

The assumption that a resource ζ_h is *non-preemptable* can be stated as:

$$\sum_{n \in \mathbb{N}, k=1..m} Z^h_{k,\,n}\left(\int_0^t \Pi_{k,n}(x)dx\right) \in \{0, 1\}. \tag{3.23}$$

3.1.3 Priority Ceiling Protocol

In this section we show how the principle that is underlying the static and dynamic versions of the *priority ceiling protocol* [9] (PCP) can be formalized in a general way. The aim is to facilitate extensions to policies other than FPP and EDF for which the PCP has initialy been designed. In Section 3.3.2 we will apply the *priority ceiling principle* to Round Robin.

The priority ceiling principle By definition, a *non-preemptable* resource can not be used while some other task has started to use it but has not yet finished. A typical example is a shared memory zone which serves as communication channel between tasks. While a task is writing into the memory zone it must not be interrupted because otherwise the memory zone might be in an inconsistent state when it is read by the preempting task. The consequences may endanger the integritiy of the system. Under non-preemptive policies, the constraints imposed by (3.23) is automatically satisfied because once an instance $\tau_{k,n}$ has started, it executes without interruption. Under preemptive scheduling policies on the other hand, it is necessary to guarantee the safe access to critical resources. *Semaphores* can keep track of the state of a resource. If an instance $\tau_{i,j}$ wants to use a resource, it first has to ask the semaphore if the resource is free. If the resource is in use, then the instance has to wait until the resource is available again.

Semaphores may have an effect on the scheduling policy. Under FPP they induce *priority inversions* [9]. Consider for example a non-preemptable resource and three tasks τ_5, τ_6 and τ_7, to be scheduled according to FPP. Suppose τ_5 has the highest and τ_7 the lowest priority among them. Let at some time, an instance $\tau_{7,n}$ enter a critical section and lock the resource. Suppose that before $\tau_{7,n}$ releases the semaphore, an instance $\tau_{5,j}$ is activated and needs the resource too. It has a higher priority than $\tau_{7,n}$ but it is blocked by the semaphore. Thus $\tau_{7,n}$ can continue to execute. This is a contradiction with the intended priority order and referred to as *priority inversion*. Furthermore, while $\tau_{5,j}$ is blocked by $\tau_{7,n}$, instances of τ_6 can preempt $\tau_{5,n}$, by preempting the critical section of $\tau_{7,n}$. Thus, priority inversions can increase the response time of $\tau_{5,j}$. It can even lead to a deadlock, see Section 3.1.4.

A solution for these problems is the *Priority Ceiling Protocol* [9]. Notice that while the higher priority instance $\tau_{5,j}$ is blocked by $\tau_{7,n}$, its priority is *de facto* decreased below the priority of $\tau_{7,n}$. Instead, one could increase the priority of the blocking instance $\tau_{7,n}$ to some level above the priority of $\tau_{5,n}$, that is, one could use some kind of priority promotion. The access to the resource would still be protected against preemption as required, but instances of τ_6 would not be able to preempt $\tau_{7,n}$ anymore. This is the basis of the *priority ceiling protocol.* Its advantages are shorter periods of priority inversion and deadlock avoidance.

Under FPP, the priority of τ_7 is increased to the *priority ceiling* associated with the semaphore that guards the resource. The priority ceiling is the highest priority of all tasks that may lock the resource. Recall that in our model, priorities are assigned to instances and not to tasks as in [9]. Here the ceiling must be a priority higher than the priority of all instances that need the resource. In our example, we could choose $\vec{Q}^h = (5)$, since $(5) \succ (5, n)$, $\forall\, n$, according to convention (2.27) on page 13.

Chen and Lin [2] have extended the PCP for EDF. It is called *dynamic* PCP, because in that case ceilings must be time dependent in order to obtain the desired properties. If an instance enters a critical section its deadline is shortened to a deadline that ensures that any other instance, needing the same resource and being released later, has a longer deadline. A ceiling is a deadline depending on the release time of the instance which enters the critical section.

From the static and the dynamic PCP arises a general principle which can be stated in a more general way as:

Definition 3.7 (Priority Ceiling Principle) *When an instance of a task enters a critical section, its priority must be promoted to a level, called* priority ceiling, *that is above the priorities of any other instance, that will need the same resource in the future. Finally, the priority is decreased to the normal level as soon as the critical section ends.*

For applying the principle to other policies, we express it in terms of the concepts of our model. For this purpose, let $\overline{\Gamma}_{k,n}(.)$ be the priority function corresponding to the policy which is modified by the PCP. We suppose that $\overline{\Gamma}_{k,n}(.)$ can only change its value when $\tau_{k,n}$ is executed:

$$\Pi_{k,n}(t) = 0, \qquad \Rightarrow \qquad \overline{\Gamma}_{k,n}(t^-) = \overline{\Gamma}_{k,n}(t). \tag{3.24}$$

Recall that priority functions are right-continuous.

The possibly time dependent priority ceiling associated with the resource ζ_h will be denoted $\vec{Q}^h(.)$. To be compliant with the principle, the ceiling must be such that if an instance $\tau_{k,n}$ that needs the resource ζ_h is released after t, then its priority must be majorized by the value of the ceiling $\vec{Q}^h(.)$ at t:

$$\forall\, \tau_{k,n} \vdash \quad t \leqslant A_{k,n}, \quad \exists\, c \vdash \; Z^h_{k,n}(c) = 1 \qquad \Rightarrow \qquad \vec{Q}^h(t) \succ \overline{\Gamma}_{k,n}(x) \quad \forall\, x \geqslant A_{k,n}. \tag{3.25}$$

This property ensures that if the priority of an instance is set to $\vec{Q}^h(t)$ at t, then it will not be preempted by an instance that will need the same resource ζ_h.

It can be noticed that (3.25) specifies only a lower bound for the priority ceiling, which means some freedom of choice. The case where ceilings are higher than any basic priority:

$$\vec{Q}^h(t) \succ \overline{\Gamma}_{k,n}(t) \quad \forall\, k,\, n,\, t,$$

is called *basic* priority ceiling protocol. In the example above a basic priority ceiling is $\vec{Q}^h(t) = (0)$. Suppose now furthermore that the tasks $\tau_1, \tau_2, \ldots, \tau_4$ do not require ζ_h. The basic ceiling induces that an instance can not be preempted during a critical section and therefore instances of $\tau_1, \ldots, \tau_4$ may be blocked. But this blocking and thus longer response times is not necessary since $\vec{Q}^h = (5)$ is also a valid ceiling and does not induce blocking of the tasks $\tau_1, \ldots, \tau_4$.

The question is then how to exactly define the priority functions of the instances. It can be noticed that an instance can only enter a critical section when it is allowed to execute, i.e. if it is currently the one with the highest priority. As it enters the critical section, its priority is immediately increased to the ceiling. This behavior is similar to PPEB, but here it concerns beginnings of critical sections. Since critical sections may be nested, a critical section may start during an ongoing other critical section. Thus, the priority is not necessarily constant in a critical section. To handle this situation, we will break instances into sub-instances at every beginning or end of a critical section. In other words a sub-instance is defined by a fixed set of required resources. A critical section appears then as a sequence of several adjacent sub-instances, see Figure 3.1.3 below. In terms of the indicator functions $Z^h_{k,n}(.)$, the execution times of the sub-instances can be defined by

$$C_{k,n,s} = \min\{c > C_{k,n,0..s-1} \mid \exists\, h \vdash Z^h_{k,n}(c^+) \neq Z^h_{k,n}(c)\},$$

where $C_{k,n,0..s} = \sum_{j=0}^{s} C_{k,n,j}$, with $C_{k,n,-1} = 0$, to simplify the notations. The activation time of the first sub-instance is $A_{k,n,0} = A_{k,n}$. In general, an instance completes its execution when it has been running for a time equal to its execution time. Since furthermore a sub-instance is ready for execution as soon as the previous one has finished, we define the activation times and execution ends by:

$$\begin{aligned} E_{k,n,s} &= \min\{t > A_{k,n,s} \mid \int_{A_{k,n,s}}^{t} \Pi_{k,n}(x)dx = C_{k,n,s}\} \\ A_{k,n,s} &= E_{k,n,s-1}. \end{aligned}$$

With each sub-instance we associate a priority function $\Gamma_{k,n,s}$ of PPEB type, which will be concatenated to obtain the priority function of the instance. Each beginning of sub-instance is due to the beginning or the end of a critical section. However the instance $\tau_{k,n}$ may be needing other resource before and after the beginning of the sub-instance. Thus, the parameters $\vec{P}_{k,n,s}$ and $\vec{Q}_{k,n,s}$ of the PPEB priority functions of a sub-instance $\tau_{k,n,s}$ must on one hand account for the ceiling of the critical section that begins, and on the other hand, it must be higher than the ceilings of the resource that continues to be used.

$$\vec{P}_{k,n,s} = \max_{\prec}\big(\{\overline{\Gamma}_{k,n}(A^{-}_{k,n,s})\} \cup \{\vec{Q}^{h}(A_{k,n}) \mid \exists\, \zeta_h \vdash Z^{h}_{k,n}(C_{k,n,0..s-1}) = Z^{h}_{k,n}(C^{+}_{k,n,0..s-1}) = 1\}\big) \quad (3.26)$$

$$\vec{Q}_{k,n,s} = \max_{\prec}\big(\{\overline{\Gamma}_{k,n}(A^{-}_{k,n,s})\} \cup \{\vec{Q}^{h}(A_{k,n}) \mid \exists\, \zeta_h \vdash Z^{h}_{k,n}(C^{+}_{k,n,0..s-1}) = 1\}\big). \quad (3.27)$$

The priority function of the initial instance is defined by

$$\Gamma_{k,n}(t) = \Gamma_{k,n,s}(t) \quad \text{for } E_{k,n,s-1} \leqslant t < E_{k,n,s}, \quad (3.28)$$

with $E_{k,n,-1} = -\infty$.

Example Consider the example of a set consisting in five tasks with three non-preemptable resources, see Figure 3.1.3. Suppose the scheduling policy is FPP with τ_1 having the highest priority: $\overline{\Gamma}_{k,n}(t) = (k, n)$. Let us determine the priority ceiling for ζ_1. At any time, the instances with the highest priority among the instances that need resource ζ_1 belong to τ_4. With $\vec{Q}^1(t) = (4)$, (3.25) is satisfied and the ceiling is just as high as necessary, since it is below the priorities of the instances of τ_3, τ_2, τ_1. Thus we choose

$$\vec{Q}^1(t) = (4) \qquad \vec{Q}^2(t) = (1) \qquad \vec{Q}^3(t) = (3).$$

Suppose that τ_5 needs the three resources as shown in Figure 3.1.3. Notice the execution times of the resulting sub-instances.

task	resource
τ_1	ζ_2
τ_2	
τ_3	ζ_2, ζ_3
τ_4	ζ_1
τ_5	$\zeta_1, \zeta_2, \zeta_3$
τ_6	ζ_3

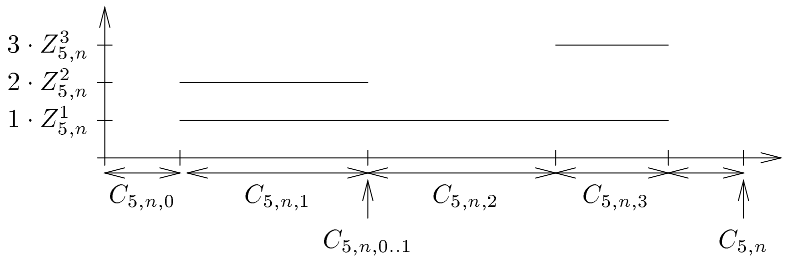


Figure 4: Sub-instances due to critical sections.

The parameters of their PPEB priority functions are, according to (3.26) and (3.27):

	$A_{5,n,0}$	$A_{5,n,1}$	$A_{5,n,2}$	$A_{5,n,3}$	$A_{5,n,4}$
$\vec{P}_{5,n,s}$	$(5,n)$	$(5,n)$	(4)	(4)	$(5,n)$
$\vec{Q}_{5,n,s}$	$(5,n)$	(1)	(4)	(3)	$(5,n)$

For example, when $\tau_{k,n,1}$ is activated, the instance does not yet use any resource. Thus, $\vec{P}_{k,n,1} = \overline{\Gamma}_{k,n}(t) = (5,n)$. On the other hand ζ_1 and ζ_2 are needed for the execution of $\tau_{k,n,1}$. Thus, $\vec{Q}_{k,n,1} = \max\big((5,n),(4),(1)\big)$.

Notice that the sub-instance $\tau_{5,n,2}$ actually has a constant priority function since it continues to need a resource which has already been locked by the instance.

Remark 3.8 In the particular case where the processing unit is considered as (unique) non-preemptive resource, an instance is a critical section from its beginning to its end. There is then only one sub-instance with $B_{k,n,0} = B_{k,n}$ and $E_{k,n,0} = E_{k,n}$. This leads to the definition of *non-preemptive* scheduling policies, see Section 3.4.

Properties The purpose of PCP priority functions is twofold. They are designed to guarantee the non-preemptive use of resources (3.23) and to avoid deadlocks. In this section we prove (3.23) and some properties which are useful when deriving response time bounds. In the framework of our model, deadlock avoidance is a direct consequence of the proof of existence of the scheduled task process (A, C, D, Π), see Section 3.1.4 for further explanations.

Proposition 3.9 *Resources are used non-preemptively* (3.23) *if the tasks that need these resources have priority functions of the form* (3.28), *based on priority ceiling functions that satisfy* (3.25).

Proof: We proceed by contradiction. Suppose there is a time t_0, where two different instances $\tau_{k,n} \neq \tau_{i,j}$ are using the same resource ζ_h:

$$Z^h_{k,n}(\textstyle\int_0^{t_0} \Pi_{k,n}(x)dx) = Z^h_{i,j}(\int_0^{t_0} \Pi_{i,j}(x)dx) = 1.$$

Let $B_{k,n,s}$ and $B_{i,j,s'}$ be respectively the execution beginnings of the sub-instances where the instances have started to used the resource:

$$B_{k,n,s} = \max\{u < t_0 \mid Z^h_{k,n}(\textstyle\int_0^{u} \Pi_{k,n}(x)dx) = 0\} \qquad B_{i,j,s'} = \max\{u < t_0 \mid Z^h_{i,j}(\int_0^{u} \Pi_{i,j}(x)dx) = 0\}.$$

An instance can only start to use a resource at some time if it is executed at that time. Thus $\Pi_{k,n}(B_{k,n,s}) = 1$ and $\Pi_{i,j}(B_{i,j,s'}) = 1$. The exclusive use condition (2.24) on page 12 implies $B_{k,n,s} \neq B_{i,j,s'}$. Suppose $B_{k,n,s} < B_{i,j,s'}$. Because $B_{i,j,s'} \leqslant t_0$ and $\tau_{k,n}$ is pending during $[B_{k,n,s}, t_0[$, $\tau_{k,n}$ is pending at $B_{i,j,s'}$. Now, two cases arise:

- $A_{i,j} \in]B_{k,n,s}, B_{i,j,s'}[$: On one hand, $\tau_{k,n}$ is using ζ_h at $A_{i,j}$. Thus

  $$\Gamma_{k,n}(A_{i,j}) = \Gamma_{k,n,s}(A_{i,j}) = \vec{Q}_{k,n,s}(A_{i,j}) \succ \vec{Q}^h(A_{k,n}).$$

  On the other hand, the priority ceiling principle implies $\vec{Q}^h(A_{k,n}) \succ \overline{\Gamma}_{i,j}(x)$, for $x \geqslant A_{k,n}$, see (3.25) on page 26. Thus, $\Gamma_{k,n}(A_{i,j}) \succ \overline{\Gamma}_{i,j}(A_{i,j})$ and therefore $\tau_{k,n} \in \mathcal{H}_{i,j}(A_{i,j})$, i.e. $\tau_{i,j}$ can not start to execute. A PCP priority function (3.28) does not change if the instance is not executed. Thus, $\Gamma_{i,j}(t)$ remains unchanged, i.e. $\Pi_{i,j}(t) = 0$ for $t \in [A_{i,j}, t_0]$, which is a contradiction.

- $A_{i,j} \leqslant B_{k,n,s}$: Since $\Pi_{k,n}(B_{k,n,s}) = 1$, $\Gamma_{i,j}(B_{k,n,s}) \prec \Gamma_{k,n}(B_{k,n,s})$. Thus, $\tau_{i,j}$ has again a lower priority and keeps it until after t_0 which implies again $\Pi_{i,j}(t_0) = 0$.

■

If the PCP is added to a policy, preemption from usually lower priority instances appears. It can easily be seen that during an interval where the priority of an instance $\tau_{k,n}$ is higher than its basic priority without interruption, it uses the processor for at most the length of a critical section:

$$\Gamma_{i,j}(t) \succ \overline{\Gamma}_{i,j}(t), \quad \forall\, t \in [t_1, t_2[\qquad \Rightarrow \qquad \exists\, \zeta_h \;\vdash\; \textstyle\int_{t_1}^{t_2} \Pi_{i,j}(x)dx \leqslant \widehat{Z}^h_k. \tag{3.29}$$

For the derivation of response time bounds, a more precise property can be obtained, based on the fact that the instance under study is pending.

Proposition 3.10 *Consider a set of tasks with PCP priority functions. Suppose an instance $\tau_{k,n}$ is pending during an interval $[t_1, t_2[$, where its priority function is $\Gamma_{k,n}(t) = \overline{\Gamma}_{k,n}(t)$ and constant. If there is another instance $\tau_{i,j}$ with $\Gamma_{i,j}(t_1) \succ \overline{\Gamma}_{k,n}(t_1) \succ \overline{\Gamma}_{i,j}(t_1)$, then at t_1 $\tau_{i,j}$ is using a resource ζ_h with a priority ceiling higher than the priority of $\tau_{k,n}$:*

$$\exists\, h \vdash \quad Z^h_{i,j}(\textstyle\int_0^{t_1} \Pi_{i,j}(x)dx) = 1, \qquad \text{and} \qquad \vec{Q}^h(A_{i,j}) \succ \overline{\Gamma}_{k,n}(t_1). \tag{3.30}$$

In $[t_1, t_2[$ the instance $\tau_{i,j}$ uses the processor during a period which is limited by its longest critical section for ζ_h:

$$\textstyle\int_{t_1}^{t_2} \Pi_{i,j}(x)dx \leqslant \widehat{Z}^h_i.$$

Proof: Since priority functions are assumed to be of PCP type, they satisfy (3.28) based on (3.26) and (3.27) and ceilings that satisfy (3.25). Thus, $\Gamma_{i,j}(t_1) \succ \overline{\Gamma}_{k,n}(t_1) \succ \overline{\Gamma}_{i,j}(t_1)$, implies that there is a resource ζ_h used by $\tau_{i,j}$ with $Z^h_{i,j}(\int_0^{t_1} \Pi_{i,j}(x)dx) = 1$. The instance $\tau_{i,j}$ may be using several resources with $\vec{Q}^h(A_{i,j}) \succ \overline{\Gamma}_{k,n}(t_1)$, i.e. may be executing several nested critical sections at t_1. But since critical sections are supposed to be properly nested (3.21), there is a longest section among them which ends after all others at $E_{i,j,s}$ for some $s \in \mathbb{N}$. Recall that $\overline{\Gamma}_{k,n}(t)$ is supposed to be constant during $[t_1, t_2[$. Because after $E_{i,j,s}$, $\tau_{i,j}$ does not use any resource with a ceiling higher than $\overline{\Gamma}_{k,n}(t_1)$, the priority of $\tau_{i,j}$ drops at $E_{i,j,s}$ below the priority of $\tau_{k,n}$: $\Gamma_{i,j}(E_{i,j,s}) \prec \overline{\Gamma}_{k,n}(E_{i,j,s}) = \Gamma_{k,n}(E_{i,j,s})$. As a result $\tau_{k,n} \in \mathcal{H}_{i,j}(E_{i,j,s})$ and thus $B_{i,j,s+1} \geqslant t_2$, implying that $\Pi_{i,j}(t) = 0$ for $t \in [E_{i,j,s}, t_2[$. Therefore

$$\textstyle\int_{t_1}^{t_2} \Pi_{i,j}(x)dx = \int_{t_1}^{E_{i,j,s}} \Pi_{i,j}(x)dx.$$

The instance $\tau_{i,j}$ has been running for $\int_0^{t_1} \Pi_{i,j}(x)dx$ units of time at the beginning of the interval and for $\int_0^{E_{i,j,s}} \Pi_{i,j}(x)dx$ units of time at its end. For any amount c between these two values, the indicator function $Z^h_{i,j}$ must is equal to one, according to (3.30):

$$c \in [\textstyle\int_0^{t_1} \Pi_{i,j}(x)dx, \int_0^{t_2} \Pi_{i,j}(x)dx[\qquad \Rightarrow \qquad Z^h_{i,j}(c) = 1.$$

Thus, the definition of the longest critical section (3.22) on page 24 implies

$$\textstyle\int_0^{t_2} \Pi_{i,j}(x)dx - \int_0^{t_1} \Pi_{i,j}(x)dx \leqslant \widehat{Z}^h_i.$$

■

Notice that the interval $[t_1, t_2[$ may be longer than $\widehat{Z}^h_i$ because the task may be preempted while keeping the lock on the resource.

Response times Under the PCP, the priority function of an instance $\tau_{k,n}$ increases and decreases depending on the priorities of the sub-instances. An exact formula of the execution end depends therefore on the exact lengths of its sub-instances, i.e. the exact lengths and positions of the critical sections. Aside from the required detailed knowledge about the critical sections, the execution end formula is also likely to become quite complex. On the other hand, the aim of the PCP is not to ensure the feasibility of tasks; the priority of an instance without critical sections is not promoted. We therefore propose to derive a bound based on Proposition 2.18 on page 17 by ignoring the priority increases, i.e. by using the basic priority function $\overline{\Gamma}_{k,n}(t)$, which is a lower bound:

$$\Gamma_{k,n}(t) \succ \overline{\Gamma}_{k,n}(t).$$

According to Proposition 2.18, it implies response time bounds larger than the actual maximum, but this is only the case if a task always ends with a critical section. If an instance does not end with a critical section, then the bound is equal to its actual execution end.

Recall equation (2.38) on page 16 for the execution end of an instance. We assume that the priority function which we use for the task under study is time independent: $\overline{\Gamma}_{k,n}(t) = \overline{\Gamma}_{k,n}(0)$. The associated set $\overline{\mathcal{H}}_{k,n}(t)$ of instances to be taken into account contains among other instances those with a larger basic priority function:

$$\mathcal{P}_{k,n} = \{\tau_{i,j} \mid \exists\, t \;\vdash\; \overline{\Gamma}_{i,j}(t) \succ \overline{\Gamma}_{k,n}(0)\}. \tag{3.31}$$

As for PPEB, it is convenient to define an interference period based on $\mathcal{P}_{k,n}$:

$$U_{k,n} \leqslant A_{k,n}, \quad W_{\mathcal{P}_{k,n}}(U_{k,n}) = 0 \qquad \text{and} \qquad W_{\mathcal{P}_{k,n}}(t) > 0 \quad \forall\, t \in]U_{k,n}, A_{k,n}]. \tag{3.32}$$

With the help of Proposition (A.4), the execution end formula (2.40) on page 17 can be proven to be equivalent to

$$E_{k,n} = \min\{t > U_{k,n} \mid C_{k,n} + \textstyle\int_{U_{k,n}}^{t} \Pi_{\overline{\mathcal{H}}_{k,n}(t)}(x)dx + U_{k,n} = t\}.$$

Consider an instance $\tau_{i,j} \in \overline{\mathcal{H}}_{k,n}(t) \setminus \mathcal{P}_{k,n}$, at some time $t \in [U_{k,n}, E_{k,n}[$. The priority of $\tau_{i,j}$ has necessarily been increased to a level above $\overline{\Gamma}_{k,n}(0)$, at the execution beginning $B_{i,j,s}$ of some sub-instance $\tau_{i,j,s}$:

$$\Gamma_{i,j}(t) \succcurlyeq \vec{Q}_{i,j,s} \succ \overline{\Gamma}_{k,n}(0) \succ \vec{P}_{i,j,s}.$$

We suppose that $B_{i,j,s}$ is the latest such time before $U_{k,n}$, where the priority changes from below to above $\overline{\Gamma}_{k,n}(0)$. Since during the interval $[U_{k,n}, E_{k,n}[$ there is always an instance with a priority higher or equal to $\overline{\Gamma}_{k,n}(0)$, $\tau_{i,j,s}$ must have started its execution before $U_{k,n}$ and therefore $\Gamma_{i,j}(U_{k,n}) \succcurlyeq \overline{\Gamma}_{k,n}(0)$. Thus, with

$$\mathcal{B}_{k,n} = \{\tau_{i,j} \notin \mathcal{P}_{k,n} \mid \Gamma_{i,j}(U_{k,n}) \succ \overline{\Gamma}_{k,n}(0)\}, \tag{3.33}$$

we have $\overline{\mathcal{H}}_{k,n}(t) = \mathcal{P}_{k,n} \cup \mathcal{B}_{k,n}$, for all $t \in [U_{k,n}, E_{k,n}[$.

Proposition 3.11 *The execution end of instances with a time independent priority function scheduled under the PCP is bounded above by*

$$\overline{E}_{k,n} = \min\{t > U_{k,n} \mid C_{k,n} + \textstyle\int_{U_{k,n}}^{t} \Pi_{\mathcal{B}_{k,n}}(x)dx + S_{\mathcal{P}_{k,n}}(U_{k,n}, t) + U_{k,n} = t\}. \tag{3.34}$$

At most one instance of the set $\mathcal{B}_{k,n}$ is pending at $U_{k,n}$:

$$\#\{\tau_{i,j} \in \mathcal{B}_{k,n} \mid \exists\, t \geqslant U_{k,n}, \;\vdash\; W_{i,j}(t) > 0\} \leqslant 1. \tag{3.35}$$

Proof: Since $\overline{\mathcal{H}}_{k,n}(t) = \mathcal{P}_{k,n} \cup \mathcal{B}_{k,n}$, the integral in the execution end formula can be decomposed. The term in $\mathcal{P}_{k,n}$ can be transformed as follows:

$$\textstyle\int_{U_{k,n}}^{t} \Pi_{\mathcal{P}_{k,n}}(x)dx = W_{\mathcal{P}_{k,n}}(U_{k,n}) + S_{\mathcal{P}_{k,n}}(U_{k,n}, t) - W_{\mathcal{P}_{k,n}}(t) \leqslant S_{\mathcal{P}_{k,n}}(U_{k,n}, t),$$

since $W_{\mathcal{P}_{k,n}}(U_{k,n}) = 0$, by the definition of $U_{k,n}$ and $W_{\mathcal{P}_{k,n}}(t) > 0$. In order to apply Proposition A.5, let

$$\begin{aligned} f(t) =& C_{k,n} + \textstyle\int_{U_{k,n}}^{t} \Pi_{\overline{\mathcal{H}}_{k,n}(t)}(x)dx \\ =& C_{k,n} + \textstyle\int_{U_{k,n}}^{t} \Pi_{\mathcal{B}_{k,n}}(x)dx + S_{\mathcal{P}_{k,n}}(U_{k,n}, t) \;\stackrel{\text{def}}{=}\; \hat{f}(t). \end{aligned}$$

We have $f(t) \leqslant \hat{f}(t)$, but $f(E_{k,n}) = \hat{f}(E_{k,n})$, since $W_{\mathcal{P}_{k,n}}(E_{k,n}) = 0$, see (2.39) on page 16. Thus, Proposition A.5 implies (3.34).

We prove now that at most one instance of $\mathcal{B}_{k,n}$ is pending at $U_{k,n}$. In the preliminary discussion above the proposition, we have seen that the priority of an instance $\tau_{i,j} \in \mathcal{B}_{k,n}$, is necessarily promoted to a level $\vec{Q}_{i,j,s}$ above $\overline{\Gamma}_{k,n}(0)$ at the execution beginning $B_{i,j,s}$ of one of its sub-instances and this before $U_{k,n}$. Suppose $W_{i,j}(U_{k,n}) > 0$ and that there is a second instance $\tau_{i',j'}$ which is also pending at $U_{k,n}$. This second instance has been promoted at some $B_{i',j',s'}$. Necessarily $\Pi_{i,j}(B_{i,j,s}) = 1$ and $\Pi_{i',j'}(B_{i',j',s'}) = 1$, but two instances can not execute at the same time, $B_{i,j,s} \neq B_{i',j',s'}$. Let us assume $B_{i,j,s} < B_{i',j',s'}$. Then, at $B_{i',j',s'}$, the priority of $\tau_{i,j}$ has already been promoted to $\vec{Q}_{i,j,s} \succcurlyeq \overline{\Gamma}_{k,n}(0)$. On the other hand, $\Gamma_{i',j',s'}(B^{-}_{i',j',s'}) = \vec{P}_{i',j',s'} \prec \overline{\Gamma}_{k,n}(0)$. Thus $\tau_{i,j}$ has a higher priority than $\tau_{i',j'}$ just before $B_{i',j',s'}$, i.e. $\tau_{i,j} \in \mathcal{H}_{i',j'}(B_{i',j',s'})$, But then, since $W_{\tau_{i,j,s}}(t) > 0$, for $t \in [B_{i,j,s}, U_{k,n}[$, $\Pi_{i',j'}(B_{i',j',s'}) = 0$, which is a contradiction. Thus, there can be only one pending instances from $\mathcal{B}_{k,n}$ at $U_{k,n}$. ■

3.1.4 Deadlocks and proof of existence

It is known that if semaphores are used to protect non-preemptable resources without the use of an appropriate protocol, then a deadlock may occur. An example is the following. Suppose the first instances $\tau_{1,0}$ and $\tau_{2,0}$ of two different tasks need two different resources, as shown in Figure 5. The instance $\tau_{1,0}$ needs first resource ζ_1 and then both, whereas the instance $\tau_{2,0}$ of the other task needs

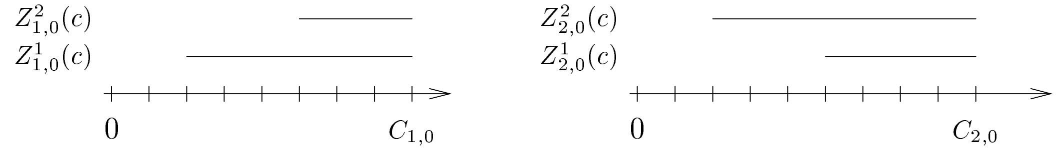


Figure 5: Two instances needing the same two resources.

first resource ζ_2 and then both. Suppose $\tau_{1,0}$ has a higher priority than $\tau_{2,0}$. If $\tau_{2,0}$ is activated first as shown in Figure 6, then it will be able to execute and to use ζ_2 after 2 units of time. At $A_{1,0}$, it is preempted by $\tau_{1,0}$ which uses ζ_1 after 2 units of time. But from t_0 on $\tau_{1,0}$ also needs ζ_2 which is locked by $\tau_{2,0}$. Then, only $\tau_{2,0}$ can continue, but after 1 time unit, $\tau_{2,0}$ does also need resource ζ_1, which is locked by $\tau_{1,0}$. This is a *deadlock* situation, since both instances need a resource which is locked by the other instance.

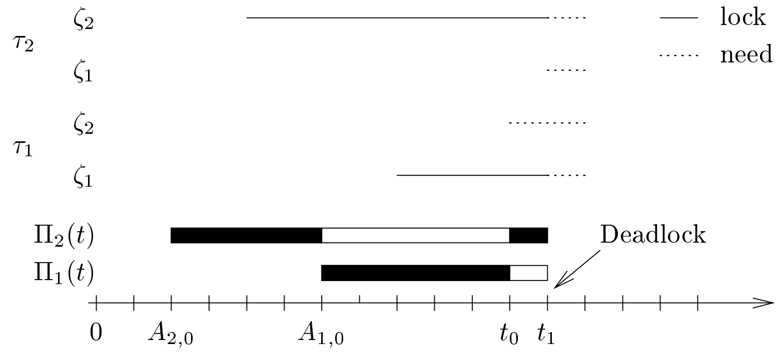


Figure 6: Deadlock on non-preemptable resources.

In [9] (and [2]) it is proven that the (dynamic) PCP prevents such situations. In our model, this is implied by the proof of existence of the scheduled task process (A, C, D, Π), see Section 3.6. Stated differently, if a scheduling policy (= priorities + semaphores) is such that a deadlock may occur, then any attempt to prove the existence of (A, C, D, Π) fails. In the deadlock example, when at t_0, $\tau_{1,0}$

stops running because it needs resource 1 which is locked by $\tau_{2,0}$, then the priority of $\tau_{1,0}$ is de facto decreased below the priority of $\tau_{2,0}$; recall the highest priority first paradigm. When at t_2, $\tau_{2,0}$ does also need resource 1, which is locked by $\tau_{1,0}$, then its priority decreases below the priority of $\tau_{1,0}$. As a result both instances should have a lower priority than the other one. This is a lack of decidability, which makes the existence of (A, C, D, Π) impossible.

The above discussion illustrates the meaning of the proof of existence: it ensures that a scheduling policy is indeed correctly realizable.

3.1.5 Locking periods

In this section, we derive a bound for the length of the periods of time during which a task locks a non-preemptable resource.

Recall the introduction of sub-instances in Section 3.1.3. An interval where a task keeps a lock on some resource ζ_h, begins with the execution beginning $B_{k,n,s}$ of some sub-instance s and ends with the execution end $E_{k,n,s'}$ of some possibly different sub-instance $s' \geqslant s$. Similarly to (2.40), we have

$$E_{k,n,s'} = \min\{t > B_{k,n,s} \mid C_{k,n,s..s'} + \textstyle\int_{B_{k,n,s}}^{t} \Pi_{\mathcal{H}_{k,n}(x)}(x)dx + B_{k,n,s} = t\}.$$

For $t \in [B_{k,n,s}, E_{k,n,s'}]$, $\Gamma_{k,n}(t) \succcurlyeq \vec{Q}^h(A_{k,n})$, i.e. the priority of $\tau_{k,n}$ is higher than or equal to the ceiling associated with the used resource. It may be higher because of a nested critical section with a ceiling higher than $\vec{Q}^h(A_{k,n})$. We therefore bound the priority function of $\tau_{k,n}$ from below by $\vec{Q}^h(A_{k,n})$. The majorizing set of higher priority instances $\mathcal{H}'_{k,n}(x) \supset \mathcal{H}_{k,n}(x)$ is then

$$\mathcal{H}'_{k,n}(x) = \{\tau_{i,j} \mid \Gamma_{i,j}(t) \succ \vec{Q}^h(A_{k,n})\}.$$

It contains among other instances those with an initial priority higher than the ceiling:

$$\mathcal{Q}^h = \{\tau_{i,j} \mid \overline{\Gamma}_{i,j}(t) \succ \vec{Q}^h(A_{k,n})\}.$$

If $\tau_{i,j} \in \mathcal{H}'_{k,n}(x) \setminus \mathcal{Q}^h$, then its priority must have been promoted above $\mathcal{Q}^h$ and this before $B_{k,n,s}$. But since $W_{\mathcal{H}'_{k,n}(B_{k,n,s})}(B_{k,n,s}) = 0$, $\tau_{i,j}$ must have completed before $B_{k,n,s}$, i.e. it is not executed during $[B_{k,n,s}, E_{k,n,s'}[$. Therefore

$$\Pi_{\mathcal{Q}^h}(x) + \Pi_{k,n}(t) = 1 \qquad t \in [B_{k,n,s}, E_{k,n,s'}[.$$

Thus, the execution end formula reduces to

$$E_{k,n,s'} = \min\{t > B_{k,n,s} \mid C_{k,n,s..s'} + S_{\mathcal{Q}^h}(B_{k,n,s}, t) + B_{k,n,s} = t\}. \tag{3.36}$$

3.2 Layered priorities

Consider a set of tasks scheduled according to the fixed preemptive priority assignment (FPP), i.e. with priorities of the form $\Gamma_{k,n}(t) = (k, n)$. Two instances of the same task will be executed in the *first in first out* order, since $n < n'$ implies $A_{k,n} \leqslant A_{k',n'}$ and $(k, n) \succ (k', n')$. Suppose now, that for the instances of the highest priority task τ_1, we would choose priorities of the form $\Gamma_{1,n}(t) = (1, -n)$. Then, the instance of τ_1 would be executed in the *last in first out* order, since $(1, n) \prec (1, n')$. However, response time bounds for lower priority tasks would not change, since the instances of τ_1 would still have a higher priority than the instance of some task $\tau_{k,n}$ with $k > 1$. This shows a kind of layer structure of the priorities, with each task being part of a layer. We can generalize this idea to layers that may contain several tasks, with some particular policy such as FIFO, LIFO, EDF, FPP, their non-preemptive versions or Round Robin (RR) inside of the layer and a global FPP policy for the layers. We say that such policies are based on *layered preemptive priorities*. Note that the combination of FPP and RR is a Posix 1003.1b compliant scheduling policy, which is an extra motivation for defining and analyzing layered priorities.

3.2.1 Layered preemptive priorities (LPP)

Suppose the set of tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$ is subdivided into separate layers

$$\lambda_l = \{\tau_k \mid \underline{m}_l \leqslant k \leqslant \overline{m}_l\},$$

where $1 \leqslant \underline{m}_l \leqslant \overline{m}_l \leqslant m$ and $\overline{m}_l + 1 = \underline{m}_{l+1}$. Assume that pending instances in a layer λ_l are executed as soon and as long as no instance in the higher priority layers $\lambda_1, \ldots, \lambda_{l-1}$ are pending. This can be realized by using for each instance $\tau_{k,n} \in \lambda_l$ the index l of the layer to which it belongs as first priority coordinate:

$$\Gamma_{k,n}(t) = (l, \ldots). \tag{3.37}$$

Indeed, because the first coordinate is the most important, recall (2.27), $\tau_{k,n}$ has a lower priority than any instance $\tau_{i,j}$ in a higher priority layer: $\tau_{i,j} \in \lambda_{l'}$ with $l' < l$.

Definition 3.12 *A priority assignment is said to be based on* Layered Preemptive Priorities *if it is equivalent (in the sense of Definition 2.16) to an assignment of the form* (3.37).

Notice that with $\Gamma'_{k,n}(t) = (0, \Gamma_{k,n}(t))$, any priority assignment can be seen as an LPP assignment with a unique layer. A more interesting case is a set of task with FPP assignment, i.e.

$$\Gamma_{k,n}(t) = (k, n).$$

It satisfies Definition 3.12 with $l = k$, that is with each task contained in a separate layer with FIFO.

Usually, one chooses for each layer a particular scheduling policy which specifies the lower order priority coordinates, as for example:

FPP: $\Gamma_{k,n}(t) = (l, k, n)$	EDF: $\Gamma_{k,n}(t) = (l, D_{k,n}, k, n)$
LIFO: $\Gamma_{k,n}(t) = (l, -A_{k,n}, k, -n)$	FIFO: $\Gamma_{k,n}(t) = (l, A_{k,n}, k, n)$

but also Round Robin, see Section 3.3.

3.2.2 Priority promotion and ceilings under layered priorities

In this section we will apply the Priority Ceiling Principle, see Section 3.1.3 to layered priorities. In order to be compatible with the overall priority structure, a ceiling must be of the form

$$\vec{Q}^h(t) = (l, \ldots). \tag{3.38}$$

The priority function that is associated with a resource depends on the policy in the layer λ_l that contains the instances $\tau_{k,n}$ with the highest priority among the instances that can use ζ_h after t.

Suppose this layer is scheduled according to FPP. Then, there is a highest priority task that needs the resource. Let this task be τ_k and let $\tau_{k,n}$ be its next instance to be released after t. It is at t, the highest priority instance among the instances that will need ζ_h in the future after t. We could therefore chose $(l, k, n-1)$ as priority ceiling. However, since earlier instances of τ_k have already completed at t it is equivalent to take

$$\vec{Q}^h(t) = (l, k) \succ (l, k, n), \tag{3.39}$$

which is preferable because it only depends on k.

Suppose now that, the layer is scheduled according to EDF. Notice that several tasks of the layer may need the resource. At any time, the highest priority instance among the instances that will need ζ_h in the future after t, is the one with the shortest absolute deadline among the instances that will need the resource. Since we can assume that the release times in the future after t are known, we use the earliest possible absolute deadline, which can be obtained from the relative deadlines of the task that need the resource:

$$\vec{Q}^h(t) = (l, t + \min\{\overline{D}_{i,j} \mid \tau_{i,j} \in \lambda_l,\ \exists\ c\ \vdash\ Z^h_{i,j}(c) = 1\}). \tag{3.40}$$

If the policy of the layer is LIFO, then at some time t, the instances that need the resource in the future all have a higher priority than the instance to be promoted at t. We must therefore choose their supremum as ceiling

$$\vec{Q}^h(t) = (l). \tag{3.41}$$

For a set of tasks scheduled under LIFO, this means that during a critical section an instance becomes non-preemptable, under PCP.

Consider the case of a layer scheduled according to FIFO. Notice that FIFO is a non-preemptive policy, i.e. once the execution of an instance has started it can not be interrupted by instances of the layer. In other words the PCP is not needed for a set of task under FIFO. In the context of layered priorities, a ceiling is however needed in the case where the FIFO layer contains the highest priority instances that need the considered resource. We could choose (l, t) which would be the priority of an instance released at t. But since at the time of promotion no instance of the layer is pending, i.e. all instance released in the past have already completed, it is equivalent to chose

$$\vec{Q}^h(t) = (l), \tag{3.42}$$

which is simpler.

3.3 Round Robin (RR)

The underlying idea of the Round Robin policy is to reserve for each task a certain rate of the available processor capacity by recurrently permitting each task to execute during a certain time span. This is not a typical real-time policy, but if response time bounds can be computed, a feasibility test can be performed. Furthermore, if tasks are also feasible under an other policy than the optimal ones, the alternative policy can be chosen in order to satisfy additional properties (regularity of service, fairness, etc). It is interesting to notice that under LPP, alternative choices in one layer have no effect on the response times in other layers.

The aim of this section is to define in our model the Round Robin scheduling policy in a layer under LPP and to give reasons for using Round Robin.

3.3.1 Definition

First we will give an informal description of the Round Robin scheduling policy that we wish to define by a priority assignment. To begin with, suppose it is the highest priority layer $\lambda_1 = \{\tau_1, \tau_2, \ldots, \tau_{\overline{m}_1}\}$ which is to be scheduled according to the Round Robin policy. The scheduler works in *Round Robin cycles* during which all tasks are inspected, in the order $\tau_1, \tau_2, \ldots, \tau_{\overline{m}_1}$. A task which is found to have pending instances, is allowed to run for at most Ψ_k units of time, otherwise nothing is done and the next task is considered.

More precisely, suppose τ_k is checked at some time t, and found to have a pending instance $\tau_{k,n}$. This instance is then executed for at most Ψ_k units of time. If $W_{k,n}(t) < \Psi_k$, then $\tau_{k,n}$ does use less than the allowed maximum Ψ_k. If the following instance is not yet pending, the scheduler passes on to τ_{k+1}. But if the following instance $\tau_{k,n+1}$ is pending, the scheduler continues with $\tau_{k,n+1}$ and any other pending instance of the task as long as the maximum Ψ_k is not reached.

As a result, each task gets repeatedly the opportunity to execute during an interval of Ψ_k units of time. If τ_k has pending instances then two such opportunities are separated by at most $\Psi^l = \sum_{\tau_i \in \lambda_l} \Psi_i$, the maximal length of a RR cycle. The time between two opportunities may be shorter if the other tasks did not used their respective Ψ_i time units.

If a lower priority layer λ_l, $l > 1$, is scheduled according to Round Robin, then the preemption from higher priority layers does suspend all the activities in the RR layer.

Having described the policy, we will now construct a priority assignment that implies the desired behavior of the scheduler. Notice first that this policy can be represented by an exclusive-use priority assignment since at each moment there is exactly one instance that is allowed to execute. Recall that the scheduler works in cycles, during which all tasks are checked for pending instances. If a task is found to have pending instances, it is allowed to run for at most Ψ_k units of time. This implies that the execution of the currently running instance must be interrupted as soon as the quantum Ψ_k is exhausted, if other instances are pending. To achieve this under the HPF paradigm, the priority function of the instance must be decreased at that moment. Furthermore, it must be allowed to continue its execution in the following RR cycle. We will realize this by a priority function of the form:

$$\Gamma_{k,n}(t) \stackrel{\mathsf{def}}{=} \big(l, P_{k,n}(t), k, n\big). \tag{3.43}$$

The first component is the index of the RR layer. The second component is the index of the RR cycle during which $\tau_{k,n}$ is allowed to execute. Recall that a larger component means a lower priority. A smaller index means thus a higher priority. If at t there are two pending instances $\tau_{k,n}$ and $\tau_{i,j}$ with $P_{k,n}(t) < P_{i,j}(t)$, than $\tau_{k,n}$ has a higher priority than $\tau_{i,j}$. In other words, $\tau_{k,n}$ will be executed in an earlier RR cycle than $\tau_{i,j}$. If $P_{k,n}(t) = P_{i,j}(t)$, then the two instances will run in the same RR cycle, in the order which is determined by the third and fourth component of the priority. We will define $P_{k,n}(.)$ such that if during the execution of $\tau_{k,n}$, the limit Ψ_k is reached at some time t_0, then $P_{k,n}(t_0) = P_{k,n}(t_0^-) + 1$. Thus, the priority of $\tau_{k,n}$ decreases because it is set from the index of the current RR cycle to the index of the next. Instances of τ_{k+1} are then allowed to run. According to the interpretation given above, $P_{k,n}(A_{k,n})$ is the index of the RR cycle in which $\tau_{k,n}$ will start to execute. Since instances of the same task are executed in the FIFO order, $P_{k,n}(A_{k,n})$ must be the index of the cycle where $\tau_{k,n-1}$ completes its execution or a later cycle, i.e. we must have $P_{k,n}(A_{k,n}) \geqslant P_{k,n-1}(E_{k,n-1})$. If $W_{k,n-1}(A_{k,n}) > 0$, then $P_{k,n}(A_{k,n}) = P_{k,n-1}(E_{k,n-1})$ is appropriate, so that $\tau_{k,n}$ can start as soon a $\tau_{k,n-1}$ completes. If on the other hand $\tau_{k,n-1}$ has completed before $\tau_{k,n}$ is released, then $\tau_{k,n}$ should be allowed to run as soon as possible, i.e. in the current (at the time $A_{k,n}$) or the next RR cycle. Therefore, we need a function that keeps track of the current RR cycle:

$$P(t) = \begin{cases} \min\{P_{k,n}(t^-) \mid \exists\, \tau_k \in \lambda_l,\ n \in \mathbb{N} \vdash\ W_{k,n}(t) > 0\} & \text{if } W_{\lambda_l}(t) > 0 \\ \\ 0 & \text{if } W_{\lambda_l}(t) = 0 \end{cases} \tag{3.44}$$

It is defined as the smallest RR cycle index $P_{k,n}(t)$ among the pending instances activated before t. Notice that $P(t)$ is left-continuous and depends only on the past before t, this will be important for the proof of existence (Section 3.6) of the RR priority functions. Furthermore, $P(t)$ is set to zero when when no instance is pending.

The definition implies that when an instance is pending at some time t, its RR cycle index must be larger or equal to $P(t)$. Thus, if $P_{k,n-1}(A_{k,n}) < P(A_{k,n})$ then the previous instance $\tau_{k,n-1}$ has completed when $\tau_{k,n}$ is released. In that case $\tau_{k,n}$ should run in the current RR cycle $P(A_{k,n})$ unless the RR scheduler has already checked τ_k which is the case if the task index of the instance, currently allowed to run, is larger than k. Thus we need a function that keeps track of that index:

$$K(t) = \begin{cases} \min\{k \mid \exists\, \tau_k \in \lambda_l,\ n \in \mathbb{N} \vdash\ W_{k,n}(t) > 0,\ P_{k,n}(t^-) = P(t)\} & \text{if } W_{\lambda_l}(t) > 0 \\ -1 & \text{if } W_{\lambda_l}(t) = 0. \end{cases} \tag{3.45}$$

Now, if $P_{k,n-1}(A_{k,n}) < P(A_{k,n})$, we can choose $P_{k,n}(A_{k,n}) = P(A_{k,n}) + \mathbb{1}_{[k<K(A_{k,n})]}$ to obtain the desired behavior.

If $P_{k,n-1}(A_{k,n}) = P(A_{k,n})$ and $K(A_{k,n}) > k$, then $\tau_{k,n-1}$ has already completed in the current cycle and we must also choose $P_{k,n}(A_{k,n}) = P(A_{k,n}) + \mathbb{1}_{[k<K(A_{k,n})]}$.

If $P_{k,n-1}(A_{k,n}) > P(A_{k,n})$, then $\tau_{k,n-1}$ has not yet completed and $\tau_{k,n}$ should start to run in the cycle where $\tau_{k,n-1}$ completes or in the following if with the completion of $\tau_{k,n-1}$, the limit Ψ_k is reached. Thus, we need to know $P_{k,n-1}(E_{k,n-1})$. For this purpose the last instance $\tau_{k,j}$, of the type discussed above, activated before $A_{k,n}$ is required:

$$u_{k,n} = \max\{j \leqslant n \mid P_{k,j-1}(A_{k,j}) < P(A_{k,j}) \text{ or } P_{k,j-1}(A_{k,j}) = P(A_{k,j}),\ k < K(A_{k,n})\}. \tag{3.46}$$

Then $\tau_{k,n-1}$ completes in or at the beginning of the cycle $P(A_{k,u_{k,n}}) + \lfloor C_{k,u_{k,n}..n-1}/\Psi_k \rfloor$. The reason is that each of the instance $\tau_{k,u_{k,n}}, \ldots, \tau_{k,n-1}$ is released before the previous instance has completed and thus the task can use all available quanta Ψ_k entirely which implies that for their execution $\lfloor C_{k,u_{k,n}..n-1}/\Psi_k \rfloor$ entire RR cycles are required. On the other hand, after $\tau_{k,n}$ has started to run its priority should be decreased as soon as the limit Ψ_k is again reached. This can be obtained with a priority function of the form

$$P_{k,n}(t) = \left\lfloor \frac{C_{k,u_{k,n}..n-1} + \int_0^t \Pi_{k,n}(x)dx}{\Psi_k} \right\rfloor + P(A_{k,u_{k,n}}) + \mathbb{1}_{[k<K(A_{k,u_{k,n}})]}. \tag{3.47}$$

Notice that with this formula $P_{k,n}(t) = P_{k,n}(A_{k,n})$ for $t \leqslant A_{k,n}$. The value of $P_{k,n}(t)$ is only needed for $t \geqslant A_{k,n}$ to perform the scheduling, but defining $P_{k,n}(.)$ this way is needed for the PCP and the non-preemptive version of Round Robin.

Notice also that the priority can only change when $\tau_{k,n}$ is actually running and therefore preemption form higher priority layers have the effect of suspending the Round Robin scheduler in the sense that all related functions remain constant while an instance of a higher priority layer is executed.

Above we have given a motivation for each of the equations that define the Round Robin priority functions that we want to consider. It must however be verified that the priority functions are well defined, i.e. if such a schedule can be realized. It follows from Theorem 3.15 in Section 3.6, that the functions $P(.), K(.)$ and $P_{k,n}(.)$ are indeed well defined. Furthermore, it means that $P_{k,n}(.)$ is right-continuous, because the resulting $\Gamma_{k,n}(.)$ are piecewise order preserving.

For the derivation of response time bounds, the following property is needed.

Proposition 3.13 *The function $P(.)$ is monotonically increasing when $W_{\lambda_l}(.)$ is strictly positive. It is furthermore a lower bound for the priority component $P_{k,n}(.)$ of any pending or just released instance:*

$$\forall\, t \in [A_{k,n}, E_{k,n}[: \quad P_{k,n}(t) \geqslant P(t). \tag{3.48}$$

Proof: From the defining equation (3.47) it can be seen that $P_{k,n}(.)$ is monotonically increasing. As a result, during an interval $]t_1, t_2[$, where $W_{\lambda_l}(t) > 0$ and no instance is released, $P(.)$ is monotonically increasing since the set of instances that contribute to $P(.)$ can only decrease - due to execution ends. By (3.44), if $W_{k,n}(t) > 0$, then $P(t) \leqslant P_{k,n}(t^-) \leqslant P_{k,n}(t)$, since furthermore $P_{k,n}(.)$ is increasing.

A time t where some instance is activated must be treated separately, because just after such a time, the set of instance that determine $P(.)$ increases; if $t = A_{k,n}$, then $W_{k,n}(t) = 0$, and $W_{k,n}(t^+) > 0$. First we prove that

$$P_{k,n}(A_{k,n}) \geqslant P(A_{k,n}). \tag{3.49}$$

If $n = u_{k,n}$, then $P_{k,n}(A_{k,n}) = P(A_{k,n}) + \mathbb{1}_{[k<K(A_{k,n})]} \geqslant P(A_{k,n})$. If $n \neq u_{k,n}$, then $u_{k,n} < n$. Notice first that

$$u_{k,n-1} = u_{k,n} \qquad \Rightarrow \qquad P_{k,n-1}(E_{k,n}) = P_{k,n}(A_{k,n}). \tag{3.50}$$

We have to distinguish two cases:

- $A_{k,u_{k,n}} = A_{k,n}$: then $W_{k,u_{k,n}}(t) = 0$ and (3.49) follows from (3.50).
- $A_{k,u_{k,n}} < A_{k,n}$: then there is an instance τ_j with $u_{k,n} < j < n$ and $A_{k,j-1} < A_{k,j} = A_{k,j+1} = \ldots = A_{k,n}$. The definition of $u_{k,n}$ implies that $W_{k,j-1}(t) > 0$. Furthermore, by (3.50), $P_{k,j-1}(A_{k,n}) > P(A_{k,n})$ and thus with (3.50), (3.49) is induced.

It remains to prove that $P(.)$ is increasing at the times where an instance is released. Since $P(.)$ is left-continuous, it means to prove that $P(A^+_{k,n}) > P(A_{k,n})$. Since there is an interval $[A_{k,n}, A_{k,n}+\epsilon[$ of positive length, where $\tau_{k,n}$ is pending and $P_{k,n}(.)$ remains constant, we have $P_{k,n}(t) = P_{k,n}(t^-) \geqslant P(A_{k,n})$. Since, $P(t) \geqslant P_{k,n}(t^-)$ it implies $P(t) \geqslant P(A_{k,n})$ for $t \in [A_{k,n}, A_{k,n}+\epsilon[$ and thus $P(A^+_{k,n}) \geqslant P(A_{k,n})$. ■

Illustration of priority functions Figure 7 shows the functions $P_{k,n}(.)$ and $P(.)$ for a sample

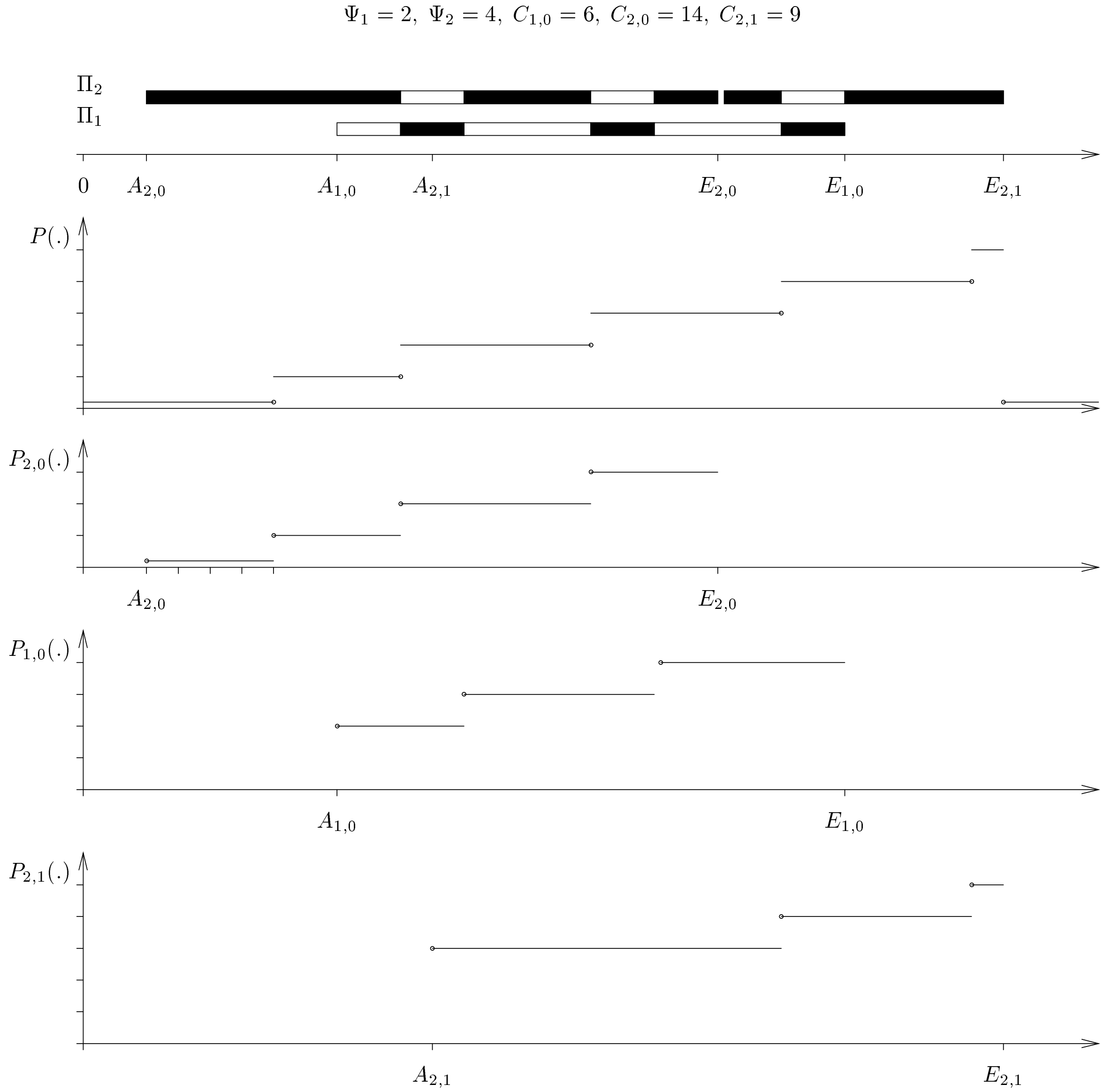


Figure 7: Round Robin: illustration of the priority components.

trajectory, where the first instance $\tau_{1,0}$ of some task τ_1 and the first two instances $\tau_{2,0}$ $\tau_{2,1}$ of some task $\tau_{2,0}$ are scheduled according to Round Robin. The functions $P_{k,n}(.)$ are only drawn where the corresponding instance is pending, whereas $P(.)$ is drawn for all times. By definition $P(t) = 0$ until the first activation of some instance, i.e for $t < A_{2,0}$.

Notice that $u_{2,0} = 0$ and thus $\tau_{2,0}$ starts with $P_{2,0}(A_{2,0}) = P(A_{2,0}) = 0$. Then, $P_{2,0}(.)$ increases as soon as $\tau_{2,0}$ has executed for $\Psi_2 = 4$ units of time. As a result, the priority decreases. However, $\tau_{2,0}$ continues to execute since it is the only pending instance.

At $A_{1,0}$ an other instance is activated. Again, $u_{1,0} = 0$, but furthermore $1 < K(A_{1,0}) = 2$, since $\tau_{2,0}$ is currently executed. Thus, $\tau_{1,0}$ starts with

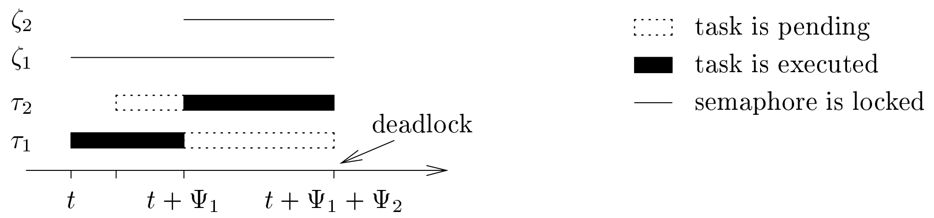
$$P_{1,0}(A_{1,0}) = \lfloor 0/\Psi_1 \rfloor + P(A_{1,0}) + \mathbb{1}_{[1<K(A_{1,0})]} = 0 + 1 + 1 = 2.$$

Notice that without the term $\mathbb{1}_{[1<K(A_{1,0})]}$, i.e. if we had $P_{1,0}(A_{1,0}) = 1$, then $\tau_{2,0}$ would immediately be interrupted, since $(l, 1, 2, 0) \prec (l, 1, 1, 0)$.

At $A_{2,1}$, the next instance of τ_2 is activated while the first one is still pending. Thus $u_{2,1} = u_{2,0} = 0$. It starts with $P_{2,1}(A_{2,1}) = \lfloor C_{2,0}/\Psi_2 \rfloor + 0 + 0 = 3$. As a result, it starts with a lower priority than $\tau_{1,0}$ and has always a lower priority than $\tau_{2,0}$. Notice that while $\tau_{2,1}$ is not allowed to execute, the function $P_{2,1}(.)$ remains constant.

3.3.2 Non-preemptable resources under Round Robin

If semaphores are used to protect non-preemptable resources, then instances may be blocked on a locked resource. As under FPP or EDF, this can result in a deadlock if nested critical sections exist. To understand how, let us consider a schematic example. Let a Round Robin layer consist in two tasks τ_1 and τ_2 with nested access to the same resources ζ_1 and ζ_2 but in a different order. Suppose τ_1 needs ζ_1 immediately and ζ_2 after Ψ_1 units of execution time. Symmetrically, τ_2 needs ζ_2 immediately and ζ_1 after Ψ_2 units of execution time. Now, if τ_1 is released at t, and τ_2 just thereafter, then τ_1 locks ζ_1 at t and is executed until $t + \Psi_1$. Then τ_2 executes and locks ζ_2. At $t + \Psi_1 + \Psi_2$, τ_1 should again be executed, but it needs ζ_2, which is locked by τ_2. Thus τ_1 is blocked. On the other hand τ_2 is also blocked since it needs ζ_1 which is locked by τ_1, i.e. a deadlock occurs.



Recall the priority ceiling principle (Section 3.1.3) which prevents from deadlocks and the property (3.25) on page 26 that must be satisfied by a priority ceiling $Q^h(.)$. Under Round Robin, priorities are decreasing with time. Thus, if at some time t_0 an instance $\tau_{k,n}$ enters a critical section where it uses a resource ζ_h, it is sufficient making it keep its priority by choosing $Q^h(t) = (l, P_{k,n}(t_0), k, n)$. It would not be preempted by an other task of the layer. On the other hand all instances with a lower priority have completed before t_0. Thus it is equivalent to choose

$$Q^h(t) = (l), \tag{3.51}$$

which is higher than any priority of the layer and preferable because independent of task instances.

To understand the effect of this priority ceiling let us consider a set of tasks entirely scheduled according to Round Robin. Consider an instance $\tau_{k,n}$ that requires only one resource. Let it start to execute a critical section at some time $B_{k,n,s}$. Its priority is then promoted to $\vec{Q}^h(A_{k,n,s}) = (-1)$ and remains constant until the end of the critical section at some time $E_{k,n,s'}$. The promotion is then canceled and the priority is set back to $\overline{\Gamma}_{k,n}(t) = (P_{k,n}(t), k, n)$. Recall that this priority depends on the amount of time the instance has actually executed. If the instance was able to executed for $3 \cdot \Psi_k$ units of time because of the critical section, then $P_{k,n}(E_{k,n,s}) = P_{k,n}(A_{k,n,s}) + 3 = P(E_{k,n,s}) + 3$. This means that the instance will not be able to execute in the following 2 RR cycles. In other words, it has to wait until the other tasks have used the processing time that is normally reserved for them by the Round Robin scheduling rule. This is a kind of automatic correction of the violated Round Robin rule. Notice that this mechanism does operate on the task as a whole. If the instance $\tau_{k,n}$ in the above example ends with a critical section then the following instance $\tau_{k,n+1}$ inherits the low priority $P(E_{k,n,s}) + 3$, i.e. will only run after two RR cycles.

3.4 Non-preemptive policies

A scheduling policy is non-preemptive if instances execute without being interrupted once they have started their execution. Among the policies that we have considered, only FIFO is non-preemptive. But by applying the concept of non-preemptable resource to the processor, non-preemptive versions of preemptive scheduling policies appear naturally. Non-preemptive priorities are used for example for the message communication on the field-bus CAN [13]. The bus can not be used preemptively. For a usual microprocessor on the other hand it is possible to choose a preemptive or non-preemptive policy. The non-preemptive case is characterized by the fact that once an instance has started to execute, it completes without interruption. This has advantages:

- The scheduling of a set of tasks on a real-world processing unit according to some scheduling policy is itself a task that is usually performed on the same processing unit. This activity means overheads which reduce the time available to the tasks. Katcher et al. [7] have identified different kinds of overheads. If an instance is preempted its state must be saved to allow it to continue its execution later, without restarting from the beginning. Response time analysis that accounts for this and other overheads can be found in [12] for FPP and in [10] for EDF.

  The advantage of non-preemptive policies is that the scheduler has to intervene less often, which reduces these overheads.

- Tasks may need resources that can only be used in an exclusive manner. It implies that while an instance is using such a resource it may not be preempted. This condition is automatically satisfied under non-preemptive scheduling policies. Preemptive policies on the other hand must be modified, as by the *Priority Ceiling Protocol* (Section 3.1.3).

- The implementation of a non-preemptive policy is simpler because during the execution of an instance the scheduler does not need to intervene.

But it has also a disadvantage:

- Timing analysis shows that a task set which is feasible under a non-preemptive policy is also feasible under a preemptive policy, but the opposite is not necessarily true. In this view, non-preemptive policies appear to be less efficient.

It can not be decided in general whether for a concrete application a preemptive or a non-preemptive policy is more appropriate.

3.4.1 Non-preemptive versions of preemptive policies

For FPP, EDF, LIFO and RR, the non-preemptive versions can be derived by considering the processor to be a non-preemptable resource. Suppose this is realized by a semaphore. An immediate question is if a deadlock could occur. The answer is that no deadlock may occur, because there are no nested critical sections, the processor being the only non-preemptable resource. However, the priority ceiling principle is useful to determine the priority functions. According to (3.25) on page 26, a ceiling must be higher than the priority of any instance released in the future, since all instances need the same non-preemptive resource. If the non-preemptive policies is to be used in a layer, then the ceiling must be a priority in the layer and larger than any priority of instances in the layer. It can be verified that under convention (2.27), the following ceilings are valid choices:

$$\begin{array}{lll} \text{NP-FP} & \vec{P}_{k,n} = (l,k,n) & \vec{Q}_{k,n} = (l) \\ \text{NP-EDF} & \vec{P}_{k,n} = (l,D_{k,n},k,n) & \vec{Q}_{k,n} = (l) \\ \text{NP-LIFO} & \vec{P}_{k,n} = (l,-A_{k,n},k,-n) & \vec{Q}_{k,n} = (l) \\ \text{NP-RR} & \vec{P}_{k,n} = (l,P_{k,n}(A_{k,n}),k,n) & \vec{Q}_{k,n} = (l). \end{array}$$

In principle, these are PCP priority functions but here every instance has only one critical section which is the instance itself. Thus, there is only one priority promotion at the execution beginning, which implies that the priority functions actually are of PPEB type.

These policies are similar by the fact that an executing instance is not interrupted (by an other instance of the layer). The difference lies in the order in which a next instance is polled among the pending instances, when an instance completes its execution. Compared to FIFO, these policies reorder the pending instances according to some criterion, within the limits of the non-idling assumption, instead of executing them in the order of their activation times. Under NP-RR, for example, the criterion is based on the quanta Ψ_k. The longer the execution time of an instance and the shorter the quantum, the longer a following pending instance is delayed, when instances of other tasks are pending.

NP-EDF is known to be optimal in the class of non-preemptive policies. But if other policies do also provide a feasible schedule of a task set, then additional criteria can be considered for selection the best policy for specific application. Which kind of properties the different policies have is an open question for future investigations.

3.4.2 Layered priorities and non-preemptive policies

In the same way as preemptive policies can be combined as preemptive layers, non-preemptive policies can be combined as non-preemptive layers. In that case an instance is not preempted by any task of the set, once it has started its execution. This can be obtained by a PPEB assignment with

$$\vec{P}_{k,n} = (l,\dots) \qquad \vec{Q}_{k,n} = (0), \tag{3.52}$$

i.e. with a priority after promotion which is higher than any other priority.

Remark 3.14 Consider a layer λ_l of tasks with NP-EDF priority functions. Under LPP, the parameters are $\vec{P}_{k,n} = (l,D_{k,n},k,n)$ and $\vec{Q}_{k,n} = (0)$, which is different from (3.52) and different from

$$\vec{P}_{k,n} = (D_{k,n},k,n) \qquad \vec{Q}_{k,n} = (0),$$

which would be used if the layer were scheduled alone on the processing unit. However, in all three cases we speak of NP-EDF. The reason is that for the tasks in λ_l all three priority assignments are equivalent (in the sense of Definition 2.16).

3.5 Posix 1003.1b compliant scheduling

Posix 1003.1b standard [5], formerly Posix.4, defines real-time extensions to Posix.1. Most of today's real-time operating systems conform, at least partially, to this standard. Posix 1003.1b specifies three scheduling policies : SCHED_RR, SCHED_FIFO and SCHED_OTHER. These policies apply on a process-by-process basis : each process runs with a particular policy and a given priority. Each process inherits its scheduling parameters from its father but may also change them at run-time.

- SCHED_FIFO : fixed preemptive priority with FIFO ordering among same priority processes.
- SCHED_RR : round-robin policy which allows processes of same priority to share the processing unit. Note that a process will not get the CPU until all higher priority ready-to-run processes are executed. The quantum value may be a system-wide constant, process specific or fixed for a given priority level.
- SCHED_OTHER is an implementation-defined scheduler. It could map onto SCHED_FIFO or SCHED_RR or also implement a classical Unix time-sharing policy. The standard merely mandates its presence and its documentation. Because it is not possible to expect the same behavior of SCHED_OTHER under all Posix compliant operating systems, it is strongly suggested not to use it if portability is a matter of concern and we will not consider it in our analysis.

Associated with each policy is a priority range. Depending on the implementation, these priority ranges may or may not overlap. Notice that a task as defined in our model, represents a recurrent activity which is either performed by repetitively launching a process or by a unique process that is running in cycles.

It can be seen that scheduling policies based on layered priorities with FPP, FIFO and RR are POSIX compliant. A Round Robin layer is a subset of tasks, with the same POSIX priority and SCHED_RR attribute. A FIFO layer is a similar. On the other hand, an FPP layer consists in a subset of tasks with SCHED_FIFO attributes and different POSIX priorities which are consecutive.

3.6 Proof of existence

In this section we prove that the scheduling policies which we have specified by time dependent priority functions do indeed define scheduling policies in the sense of Definition 2.3 on page 8. As pointed out in Section 3.1.4, this proof guarantees for example that the scheduler will not run into a deadlock, while trying to determine the instance with the highest priority.

Theorem 3.15
Let $\mathcal{T} = \lambda_1 \cup \ldots \cup \lambda_L$ be a set of tasks partitioned into layers and provided with an LPP priority assignment (3.37). *Inside of a layer, the priority assignment may either be time independent or Round Robin* (3.43). *Furthermore, some instances may have a PPEB priority function* (3.1) *with initial priority given by the policy of the layer to which they belong.*

In this case, there exists a unique scheduled task process so that the HPF rule is satisfied with the priority functions that correspond to the scheduling policies.

Proof: To prove this theorem, we show that it is possible to construct by recurrence a Π and a Γ, satisfying (2.31) in Lemma 2.14 on page 13 and such that Γ has the properties required by the definitions of the layer-wide scheduling policies: (3.37) on page 33 for the layer structure, (2.32) on page 15, for time independence, (3.43) on page 35 for Round Robin and (3.1)-(3.5) on page 20 for Priority Promotion.

It can be noticed that under any of the considered policies, the priority function of an instance $\tau_{k,n}$ is constant before its activation time $A_{k,n}$. Since the values of $\Gamma_{k,n}(t)$ for $t < A_{k,n}$, are not needed to define Π, we will verify only from $A_{k,n}$ on that $\Gamma_{k,n}(t)$ is well defined on $[0, A_{k,n}[$.
The construction is based on a sequence of points $\{t_h\}$, starting with

$$t_0 = \min\{A_{k,0} \mid k = 1..m\},$$

the first time where some instance is activated. Since no instance is released before t_0, we define $\Pi_{k,n}(t) = 0$, for all $t \in [0, t_0[$, $k = 1..m$, $n \in \mathbb{N}$. Because there is no activation in $[0, t_0[$, $W_{k,n}(t^+) = 0$, for all $t \in [0, t_0[$, $k = 1..m$, $n \in \mathbb{N}$ and thus (2.31) holds independently of the $\Gamma_{k,n}(t)$ which we will define later.

For each RR layer λ_l, we have to define $P(t)$ and $K(t)$. Since $W_{\lambda_l}(t) = 0$ for $t < t_0$, we set $P(t) = 0$ and $K(t) = -1$ according to (3.44) and (3.45) and hence they are well defined for $t < t_0$. For the other policies nothing need to be verified at the current step, since not priority function has yet been defined.

Now, assume that Π is defined on $[0, t_h[$ and that the $\Gamma_{k,n}$ of the instances activated before t_h are also defined on $[0, t_h[$. Suppose furthermore that (2.31) holds and that the $\Gamma_{k,n}$ with $A_{k,n} < t_h$ are defined and satisfy the properties required by the different scheduling policies. Furthermore, we suppose that Γ is constant on $[t_{h-1}, t_h[$.

Notice that since $W_{k,n}(t^+) = S_{k,n}(0, t^+) - \int_0^t \Pi_{k,n}(u)du$, the workload $W_{k,n}(t^+)$ is defined on $[0, t_h]$, i.e. including t_h. The reason is that the value of $\Pi_{k,n}$ at t_h, which is not yet defined, has no effect on the value of the integral, a point being a set of Lebesgue measure zero. Also the $P(t)$ and $K(t)$ of the RR layers are defined until t_h included, since their values depend on the past strictly before t_h.

We will define now Γ and Π until before the next construction point t_{h+1}, which has to be determined for each of the different cases that arise:

(i) $W_{1..m}(t_h^+) > 0$: For any instance activated at t_h, we first have to define $\Gamma_{k,n}(t)$ for $t < t_h$. If $\tau_{k,n}$ has a time independent priority function then it is simply $\Gamma_{k,n}(t) = \Gamma_{k,n}(0)$, for $t < t_h$. In the case of PPEB it is $\Gamma_{k,n}(t) = \overline{\Gamma}_{k,n}(t)$, for $t < t_h$. Under RR, it is

$$\Gamma_{k,n}(t) = \big(l, \lfloor C_{k,u_{k,n}..n-1}/\Psi_k \rfloor + P(U_{k,n}) + \mathbb{1}_{[k<K(U_{k,n})]}, k, n\big)$$

for $t < t_h$. Notice that since the scheduled task process is supposed to be defined until before t_h, $U_{k,n}$ is well defined. In particular, if $W_{1..m}(t_h) = 0$, then $U_{k,n} = t_h$.

Now, all instances which are pending at t_h have a priority function defined at least until before t_h. Among these instances let $\tau_{k,n}$ be the one with the highest priority just before t_h, i.e. for $\tau_{i,j} \neq \tau_{k,n}$:

$$W_{i,j}(t_h^+) > 0 \Rightarrow \Gamma_{k,n}(t_{h-1}) \succ \Gamma_{i,j}(t_{h-1}),$$

since we have assumed $\Gamma_{k,n}(t) = \Gamma_{k,n}(t_{h-1})$ for $t \in [t_{h-1}, t_h[$. Notice that $\tau_{k,n}$ can be determined because Γ is supposed to be decidable until before t_h. For the considered policies the priority of an instance does not change if it is not executed. Thus, a pending instance $\tau_{i,j} \neq \tau_{k,n}$ will continue to have a lower priority. It implies that $\tau_{k,n}$ remains the pending instance with the highest priority (at least) as long as its workload is positive, its priority does not decrease and no instance is newly activated. Thus, $\tau_{k,n}$ is the instance allowed to execute during some time span after t_h. The priority decrease concerns the case where $\tau_{k,n}$ is scheduled under RR. Because of this case let $r_h = \min\{t > 0 \mid C_{k,u_{k,n}..n-1} + \int_0^t \Pi_{k,n}(x)dx \equiv 0 \mod \Psi_k\}$ be the time where the priority decreases. If $\tau_{k,n}$ is not scheduled under RR, let $r_h = \infty$. Hence $\tau_{k,n}$ can be executed until

$$t_{h+1} = \min\big\{t_h + W_{k,n}(t_h^+),\ t_h + r_h,\ \min\{A_{i,j} > t_h \mid i = 1..m,\ n \in \mathbb{N}\}\big\},$$

that is we can define $\Pi_{k,n}(t) = 1$ and $\Pi_{i,j}(t) = 0$ for $t \in [t_h, t_{h+1}[$.

It remains to define Γ on $[t_h, t_{h+1}[$. For the instances which are not executed and for which $A_{i,j} \leqslant t_h$, let $\Gamma_{i,j}(t) = \Gamma_{i,j}(t_{h-1})$ for $t \in [t_h, t_{h+1}[$, since their priority does not change, whether $\Gamma_{i,j}$ is supposed to be time independent, or of RR or PPEB type. For $\tau_{k,n}$ on the other hand, we have to distinguish between the possible policies.

- time independent: simply set $\Gamma_{k,n}(t) = \Gamma_{k,n}(0)$.
- RR: Since $\Pi_{k,n}(t) = 1$, for $t \in [t_h, t_{h+1}[$, we can properly define

$$P_{k,n}(t) = \lfloor (C_{k,u_{k,n}..n-1} + \textstyle\int_0^t \Pi_{k,n}(x)dx)/\Psi_k \rfloor + P(U_{k,n}) + \mathbb{1}_{[k<K(U_{k,n})]}.$$

Notice that $P_{k,n}(t)$, and therefore $\Gamma_{k,n}(t)$ is constant on this interval because $t_{h+1} \leqslant r_h$.

By continuity, the value of $\int_0^{t_{h+1}} \Pi_{k,n}(x)dx$ is actually known. Thus, we can define $P_{k,n}(t)$ by the above formula for $t \in [t_h, t_{h+1}]$, i.e. until t_{h+1} included. As a result, $P(t)$ is also defined until t_{h+1} included, as required be the assumptions of the present proof by recurrence.

– PPEB: Let $\Gamma_{k,n}(t) = \vec{Q}_{k,n}$, since under PPEB instances execute at their promoted priority.

With Π and Γ defined above, (2.31) holds, because during $[t_h, t_{h+1}[$, $\tau_{k,n}$ is the pending instance with the highest priority.

For PPEB we have to verify that (3.1) holds on $[t_h, t_{h+1}[$. An instance $\tau_{i,j} \neq \tau_{k,n}$ can not start to execute at t_h. Thus, by keeping the same value as before t_h, its priority function satisfies indeed (3.1). If $\Gamma_{k,n}(t_{h-1}) = P_{k,n}$, then $\int_0^{t_h} \Pi_{k,n}(x)dx = 0$, i.e $\tau_{k,n}$ has not yet started to execute, because (3.1) is assumed to hold until before t_h. Since for $t \in [t_h, t_{h+1}[$, $\Gamma_{k,n}(t) = \vec{Q}_{k,n}$ and $\Pi_{k,n}(t) = 1$, we have $t_h = B_{k,n}$ and hence (3.1) holds on $[t_h, t_{h+1}[$. If $\Gamma_{k,n}(t^-) = \vec{Q}_{k,n}$, then $B_{k,n} < t_h$ and (3.1) also holds, since $\Gamma_{k,n}(t) = \vec{Q}_{k,n}$ for $t \in [t_h, t_{h+1}[$. Notice that (3.5) on page 20 holds in the case where $\tau_{k,n}$ has not been executed before $t_h = B_{k,n}$, since $\tau_{i,j} \in \mathcal{H}_{k,n}(t_h^-)$.

(ii) $W_{1..m}(t_h^+) = 0$: no instance is pending. Hence let

$$t_{h+1} = \min\{A_{i,j} > t_h \mid k = 1..m,\ n \in \mathbb{N}\}$$

be the next time where an instance is activated. In the interval $[t_h, t_{h+1}[$ there is then no instance pending and therefore no instance to be executed:

$$\Pi_{k,n}(t) = 0 \quad t \in [t_h, t_{h+1}[,\ k = 1..m,\ n \in \mathbb{N}.$$

Since no instance is executed the priority functions do not change their values. Thus, let

$$\Gamma_{k,n}(t) = \Gamma_{k,n}(t_{h-1}) \quad t \in [t_h, t_{h+1}[,\ k = 1..m,\ n \in \mathbb{N}.$$

Now, (2.31) trivially holds since $W_{k,n}(t) = 0$ for all instances. Furthermore, because no instance can start to execute at t_h, also (3.1) holds for instances with PPEB priority functions.

So far we have proven that the sequence of construction points is strictly increasing: $t_h < t_{h+1}$. It can be verified that each construction point t_h is either equal to an activation time, an execution end of an instance of a priority decreases under RR. By (2.2) on page 6, in an interval of finite length, the number of activations is finite. The number of pending instances and thus of execution ends is finite by (2.17). Furthermore, the priority of an instance $\tau_{k,n}$ is decreased as much as $\lceil C_{k,n}/\Psi_k \rceil$ times, which is finite. Thus, an interval of finite length contains only a finite number of construction points and thus the sequence must diverge.

In all three cases, the $\Pi_{k,n}(t)$ are defined constant on closed-open intervals $[t_h, t_{h+1}[$, that cover $\mathbb{R}_+$. Thus, each $\Pi_{k,n}$ is right-continuous.

Above we have proven the existence of Π and Γ. The uniqueness of Π is a direct consequence of Theorem 2.15. ■

4 Response time bounds

4.1 Priority promotion at execution beginning

In this section we give a Lemma that tells how to derive response time bounds when some instances have PPEB priority functions. It is in a generic form, which is valid regardless of tasks types and the values of the parameters $\vec{P}_{k,n}$ and $\vec{Q}_{k,n}$. It is a useful guideline when deriving bounds in particular cases.

4.1.1 Generic bounds

The context is a set of tasks with PPEB priority functions. We suppose that a bound for the execution times of any instance that has a lower initial priority than $\tau_{k,n}$ but a promoted priority, higher than the initial priority of $\tau_{k,n}$ is known:

$$\widetilde{Z}_k \geqslant \sup_{\omega\in\Omega}\{C_{i,j}(\omega) \mid P_{i,j}(\omega) \prec P_{k,n}(\omega),\ Q_{i,j}(\omega) \succ P_{k,n}(\omega)\}. \tag{4.1}$$

It will serve as bound for $W_{\mathcal{B}_{k,n}}(U_{k,n})$. Furthermore we suppose the existence of a set of increasing functions $\mathcal{P}_k = \{\widetilde{S}_k^B(x,a,q),\ \widetilde{S}_k^E(x,a,b,q) \mid q \in Q\}$ such that $\forall\ \omega \in \Omega, \forall n \in \mathbb{N}\ \exists q \in Q$:

$$S_{\mathcal{P}_{k,n}}(U_{k,n}, t^+;\omega) \leqslant \widetilde{S}_k^B(t-U_{k,n}, A_{k,n}-U_{k,n}, q) \tag{4.2}$$

$$S_{\mathcal{M}_{k,n}}(U_{k,n}, B_{k,n}^+;\omega) + S_{\mathcal{Q}_{k,n}}(U_{k,n}, t;\omega) \leqslant \widetilde{S}_k^E(t-U_{k,n}, A_{k,n}-U_{k,n}, B_{k,n}-U_{k,n}, q). \tag{4.3}$$

Based on there bounds we define some kind of virtual execution beginning and execution end

$$\widetilde{B}_k(a,q) \stackrel{\mathsf{def}}{=} \min\{x>0 \mid \tilde{Z}_k + \widetilde{S}_k^B(x,a,q) = x\} \tag{4.4}$$

$$\widetilde{E}_k(a,q) \stackrel{\mathsf{def}}{=} \min\{x>0 \mid \tilde{Z}_k + \widetilde{S}_k^E(x,a,\widetilde{B}_k(a,q),q) = x\} \tag{4.5}$$

which will be considered for a in

$$\mathcal{A}_k(q) \stackrel{\mathsf{def}}{=} \{a>0 \mid \widetilde{S}_k^B(x,a,q) > x,\ \forall x \leqslant a\}.$$

In terms of these quantities, a response time bound can be stated as follows

Lemma 4.1 *Let a set of tasks be scheduled with PPEB priority functions. A bound on the response times of a task τ_k is then given by*

$$R_{k,n} \leqslant \widetilde{R}_k \stackrel{\mathsf{def}}{=} \max_{q\in Q}\ \max_{a\in\mathcal{A}_k(q)} \widetilde{E}_k(a,q) - a,$$

where $\widetilde{E}_k(a,q)$ is given by (4.5).

Proof: First, we prove that $W_{\mathcal{B}_{k,n}}(U_{k,n}) \leqslant \tilde{Z}_k$. Because of (3.16), there is an instance $\tau_{i,j} \in \mathcal{B}_{k,n}$ such that $W_{\mathcal{B}_{k,n}}(U_{k,n}) = W_{i,j}(U_{k,n})$. Furthermore, by the definition of $\mathcal{B}_{k,n}$, $\vec{P}_{i,j} \prec \vec{P}_{k,n}$ and $\vec{Q}_{i,j} \succ \vec{P}_{k,n}$. Thus, the execution time $C_{i,j}$ is taken into account in the supremum in (4.1) and $W_{\mathcal{B}_{k,n}}(U_{k,n}) \leqslant \tilde{Z}_k$ is proven.

Consider an instance $\tau_{k,n}$ on a trajectory ω and let q be determined such that (4.2) and (4.3) do hold. We have $A_{k,n} - U_{k,n} \in \mathcal{A}_k(q)$, because by the definition of $U_{k,n}$, see (3.7), and (4.2) :

$$0 < W_{\mathcal{P}_{k,n}}(x) = S_{\mathcal{P}_{k,n}}(U_{k,n}, U_{k,n}+x) - \int_{U_{k,n}}^{U_{k,n}+x} \Pi_{\mathcal{P}_{k,n}}(t)dt$$

and since $1 = \Pi_{\mathcal{P}_{k,n}}(t) + \Pi_{\mathcal{B}_{k,n}}(t)$ for $t \in [U_{k,n}, A_{k,n}[$,

$$\begin{aligned} &= S_{\mathcal{P}_{k,n}}(U_{k,n}, U_{k,n}+x) - \int_{U_{k,n}}^{U_{k,n}+x} \big(1 - \Pi_{\mathcal{B}_{k,n}}(t)\big)dt \\ &\leqslant S_{\mathcal{P}_{k,n}}(U_{k,n}, U_{k,n}+x) + W_{\mathcal{B}_{k,n}}(x) - x \\ &\leqslant \widetilde{S}_k^B(x, A_{k,n}-U_{k,n}, q) + \tilde{Z}_k - x, \end{aligned}$$

for all $x \leqslant A_{k,n} - U_{k,n}$. Thus, $a = A_{k,n} - U_{k,n} \in \mathcal{A}_k(q)$. Now we prove that

$$B_{k,n} - U_{k,n} \leqslant \widetilde{B}_k(A_{k,n} - U_{k,n}, q). \tag{4.6}$$

Consider Proposition A.4 with $x_0 = U_{k,n}$,

$$f(x_0 + x) = W_{\mathcal{B}_{k,n}}(U_{k,n};\omega) + S_{\mathcal{P}_{k,n}}(U_{k,n}, t^+;\omega),$$

$\hat{x}_0 = 0$ and $\widehat{f}(\hat{x}_0 + x) = \widetilde{S}_k^B(x, A_{k,n} - U_{k,n}, q)$. Then (A.6) holds because of (4.2) and thus

$$x^* = B_{k,n} - U_{k,n} \leqslant \hat{x}^* = \widetilde{B}_k(A_{k,n} - U_{k,n}, q).$$

Then, we prove that $E_{k,n} - U_{k,n} \leqslant \widetilde{E}_k(A_{k,n} - U_{k,n}, q)$. Consider again Proposition A.4 but with $x_0 = U_{k,n}$,

$$f(x_0 + x) = W_{\mathcal{B}_{k,n}}(U_{k,n};\omega) + S_{\mathcal{M}_{k,n}}(U_{k,n}, B_{k,n}^+;\omega) + S_{\mathcal{Q}_{k,n}}(U_{k,n}, t;\omega),$$

$\hat{x}_0 = 0$ and $\widehat{f}(\hat{x}_0 + x) = \widetilde{S}_k^E(x, A_{k,n} - U_{k,n}, \widetilde{B}_k(A_{k,n} - U_{k,n}, q), q) + \tilde{Z}_k$. Then (A.6) holds because of (4.3) and (4.6). Thus

$$x^* = E_{k,n} - U_{k,n} \leqslant \hat{x}^* = \widetilde{E}_k(A_{k,n} - U_{k,n}, q),$$

and finally, by subtracting $A_{k,n} - U_{k,n}$ from both sides, we obtain

$$\widetilde{R}_{k,n} \leqslant \widetilde{E}_k(A_{k,n} - U_{k,n}, q) - (A_{k,n} + U_{k,n}) \leqslant \max_{a \in \mathcal{A}_k(q)} \widetilde{E}_k(a, q) - a \leqslant \max_{q \in Q} \max_{a \in \mathcal{A}_k(q)} \widetilde{E}_k(a, q) - a.$$

■

4.1.2 Non-preemptive policies

In this section we will derive response time bounds under non-preemptive policies, with the help of Lemma 4.1. But first, we have to introduce a particular kind of WAF.

It can be noticed, that $\tau_{k,n} \in \mathcal{M}_{k,n}$ but $\tau_{k,n} \notin \mathcal{P}_{k,n}$. In order to be able to define an accurate MWAF's for $S_{\mathcal{P}_{k,n}}$, we introduce

$$S_k^P(t_1, t_2) = \sum_{n \in \mathbb{N}} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n+1} < t_2]}. \tag{4.7}$$

This WAF does not account for the last activation in $[t_1, t_2[$. A corresponding family of MWAF must satisfy $\forall\, \omega \in \Omega, \forall\, u \in \mathbb{R}_+,\ \exists\, q \in Q$:

$$S_k^P(u, u + x;\omega) \leqslant \widehat{S}_k^P(x, q) \quad \forall\, x > 0.$$

Depending on the type of a task it may be easy to derive the $\widetilde{S}_k^P(x, q)$ from the usual MWAF's. It is the case, if for all ω, k, n there exists q such that

$$C_{k,n..n+i} \leqslant \widehat{C}_{k,0..i}(q) \qquad A_{k,n+i} - A_{k,n} \geqslant \hat{A}_{k,i}(q) \qquad \forall\, i. \tag{4.8}$$

This property means that the amount of work of any i consecutive instances is majorized by the first i activations of the MWAF. Thus, if (4.8) is true for $\tau_{k,n+i}$ and $\hat{\tau}_{k,i}$, then it is also true for $\tau_{k,n+i-1}$ and $\hat{\tau}_{k,i-1}$. In that case we can simply choose

$$\widehat{S}_k^P(x, q) = \sum_{n \in \mathbb{N}} \widehat{C}_{k,n}(q) \cdot \mathbb{1}_{[\hat{A}_{k,n} < x]} \cdot \mathbb{1}_{[\hat{A}_{k,n+1} < x]}. \tag{4.9}$$

Indeed, consider $S_k^P(u, u + x)$ and suppose that $A_{k,n-1} < u \leqslant A_{k,n}$ and $A_{k,n+i} < u + x \leqslant A_{k,n+i+1}$:

$$S_k^P(u, u + x) = \sum_{j \in \mathbb{N}} C_{k,j} \cdot \mathbb{1}_{[u \leqslant A_{k,j} < u+x]} \cdot \mathbb{1}_{[u \leqslant A_{k,j+1} < u+x]} = \sum_{j=n}^{n+i} C_{k,j} \leqslant \sum_{j=0}^{i} \widehat{C}_{k,j}$$

and since $x \geqslant A_{k,n+i+1} - A_{k,n} \geqslant \hat{A}_{k,i+1}$, because of (4.8)

$$= \sum_{j=0}^{i} \widehat{C}_{k,j} \cdot \mathbb{1}_{[\hat{A}_{k,j}<x]} \cdot \mathbb{1}_{[\hat{A}_{k,j+1}<x]}$$
$$\leqslant \sum_{j\in\mathbb{N}}^{i} \widehat{C}_{k,j} \cdot \mathbb{1}_{[\hat{A}_{k,j}<x]} \cdot \mathbb{1}_{[\hat{A}_{k,j+1}<x]} = \widehat{S}_k^P(x,q).$$

Property (4.8) is true for sporadic tasks, because the usually chosen MWAF's with

$$\widehat{C}_{k,j} = C_k \stackrel{\mathsf{def}}{=} \max_{n\in\mathbb{N},\ \omega\in\Omega} C_{k,n}(\omega) \geqslant C_{k,n} \qquad \widehat{T}_{k,j} = T_k \stackrel{\mathsf{def}}{=} \min_{n\in\mathbb{N},\ \omega\in\Omega} T_{k,n}(\omega) \leqslant T_{k,n},$$

satisfies (4.8). The two kinds of MWAF's are

$$\widehat{S}_k(x) = \big(\lceil x/T_k \rceil\big) \cdot C_k, \qquad \widehat{S}_k^P(x) = \big(\lceil \max(0,\, x - T_k)/T_k \rceil\big) \cdot C_k.$$

The same is true more generally for sporadically periodic multiframe tasks (see [8] for the definition of this type).

If (4.8) does not hold, then it is more difficult to determine $\widehat{S}_k^P(x,q)$. Suppose for example that for a sporadic task with $T_{k,n} \geqslant 2$ and $C_{k,n} \leqslant 1$ we would choose a MWAF with $\widehat{T}_{k,j} = 4$, $\widehat{C}_{k,j} = 2$ and $\widehat{A}_{k,0} = 0$. Figure 8 shows the MWAF and the WAF on some trajectory after some time u, with $A_{k,n}$ being the first instance of τ_k activated after u. It can be seen that for $\tau_{k,n+2}$, $C_{k,n..n+2} \leqslant \widehat{C}_{k,0..2}(q)$ holds but not $A_{k,n+2} - A_{k,n} \geqslant \hat{A}_{k,2}(q)$ and thus (4.8) is not satisfied. In this example we have deliberately

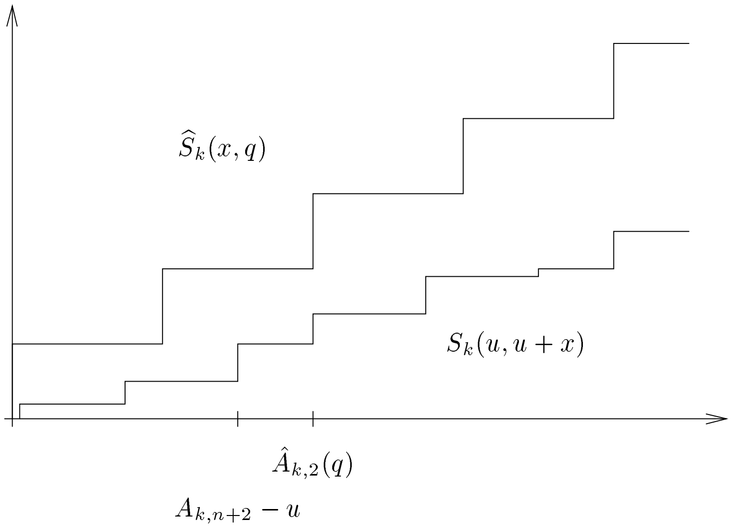


Figure 8: A general MWAF

chosen a MWAF which does not satisfy (4.8). For tasks with *mode change* MWAF's that satisfy (4.8) are not known or may not exist. The reason is that they are obtained by taking the maximum over several different WAF's [8].

The question is then how to define accurate MWAF's if (4.8) does not hold. One could simply chose $\widehat{S}_k^P = \widehat{S}_k$, which is a conservative bound. To improve this one could think of transforming the MWAF's $\widehat{S}_k$, in order to obtain some useful property similar to (4.8). To obtain (4.8) is likely to be difficult in general, since it concerns every trajectory of the task process. On the other hand, if (4.10) below, holds then (4.9) is also a valid choice and the transformation can easily be performed by inserting additional activations with $\widehat{C}_{k,j} = 0$, after activations that do not satisfy (4.10).

Proposition 4.2 *If the family of MWAF of a task satisfies*

$$\forall\, j \in \mathbb{N},\ \forall\, q \in Q \quad \widehat{T}_{k,j}(q) \leqslant \min_{n\in\mathbb{N},\ \omega\in\Omega} T_{k,n}(\omega), \tag{4.10}$$

then $\widehat{S}_k^P(x,q)$ *given by* (4.9) *is a MWAF for* $S_k^P(x,q)$.

Proof: We proceed by contradiction. Suppose that

$$S_k(u,u+x) \leqslant \widehat{S}_k(x,q) \quad \forall\, x$$

and that $A_{k,n}$ is the first activation of τ_k after or at u. Suppose furthermore that

$$S_k^P(u,u+x_0) \geqslant \widehat{S}_k^P(x_0,q), \tag{4.11}$$

for some $x_0 > 0$ and that $\widehat{S}_k^P$ is given by (4.9). Let $A_{k,n+i}$ be the last activation before or at $u + x_0$ on ω and $\hat{A}_{k,j}$ the last activation before or at x_0 for $\widehat{S}_k$:

$$A_{k,n+i} < u + x_0 \leqslant A_{k,n+i+1} \qquad \hat{A}_{k,j} < x_0 \leqslant \hat{A}_{k,j+1}.$$

Then (4.11) implies $C_{k,n..n+i-1} = S_k^P(u,u+x_0) \geqslant \widehat{S}_k^P(x_0,q) = \widehat{C}_{k,0..j-1}$. It implies $\hat{A}_{k,j} < A_{k,n+i-1} - u$, otherwise $\widehat{S}_k$ could not be majorizing. Furthermore, by the definition of $\hat{A}_{k,j}$, $\hat{A}_{k,j+1} \geqslant x_0 \geqslant A_{k,n+i-1} - u$. Thus

$$\widehat{T}_{k,j} = \hat{A}_{k,j+1} - \hat{A}_{k,j} \geqslant A_{k,n+i-1} - u - \hat{A}_{k,j}, > A_{k,n+i} - A_{k,n+i-1},$$

which is a contradiction with (4.10). ■

In the following, we suppose a layered structure of PPEB priority functions, i.e we suppose that $\vec{P}_{k,n} = (l,\dots)$ and $\vec{Q}_{k,n} = (l,\dots)$.

NPFP Let a task τ_k be part of a layer λ_l, scheduled according to NPFP. In this case $\mathcal{P}_{k,n}$ consists in all previous instances of τ_k and all instances of higher priority layers:

$$S_{\mathcal{P}_{k,n}}(U_{k,n},t^+) = S_{k,0..n-1}(U_{k,n},t^+) + S_{1..k-1}(U_{k,n},t^+).$$

We have, $S_{k,0..n-1}(U_{k,n},t^+) \leqslant \sum_{i=0}^{n-1} C_{k,i}\cdot \mathbb{1}_{[U_{k,n}\leqslant A_{k,i}]}$. On the other hand, since $A_{k,n-1} \leqslant A_{k,n}$, $S_k^P(U_{k,n},A_{k,n}^+)$ accounts for the activations after $U_{k,n}$, until at least $C_{k,n-1}$ included, i.e.

$$S_k^P(U_{k,n},A_{k,n}^+) \geqslant \textstyle\sum_{i=0}^{n-1} C_{k,i}\cdot \mathbb{1}_{[U_{k,n}\leqslant A_{n,i}]}.$$

Thus, $S_{k,0..n-1}(U_{k,n},t^+) \leqslant S_k^P(U_{k,n},A_{k,n}^+)$. Since $S_{k,0..n-1}$ is also bounded by S_k, we finally have

$$S_{k,0..n}(U_{k,n},t^+) \leqslant \min\big(S_k(U_{k,n},t^+),\ S_k^P(U_{k,n},A_{k,n}^+)\big).$$

Notice that $S_{k,0..n}(U_{k,n},t^+) \leqslant \min\big(S_k(U_{k,n},t^+),\ S_k(U_{k,n},A_{k,n}^+)\big)$ is also true, but would be a more conservative bound since $S_k^P(U_{k,n},A_{k,n}^+) \leqslant S_k(U_{k,n},A_{k,n}^+)$.

Given a family of MWAF's of the form $\mathcal{S} = \{\widehat{S}_i(x,q), \widehat{S}_k^P(x,q) \mid i = 1,\dots,\overline{m}_l,\ q\in Q\}$, a family of MWAF's as required by Lemma 4.1 consists in

$$\widetilde{S}_k^B(x,a,q) = \min\big(\widehat{S}_k(x^+,q), \widehat{S}_k^P(a^+,q)\big) + \widehat{S}_{1..k-1}(x,q) \tag{4.12}$$

$$\widetilde{S}_k^E(x,a,b,q) = \widehat{S}_k(a^+,q) + \widehat{S}_{1..\overline{m}_{l-1}}(x,q) + S_{\underline{m}_l..k-1}(b,q), \tag{4.13}$$

since $\mathcal{Q}_{k,n} = \{\tau_{i,j} \mid i \leqslant \overline{m}_{l-1}\}$. Because for all $\tau_{i,j}\in\lambda_l$ $Q_{i,j} = (l,0)$ and $P_{i,j} = (l,i,j)$, the bound (4.1) on the blocking periods from lower priority tasks is

$$\widetilde{Z}_k = \max\{C_{i,j} \mid k < i \leqslant \overline{m}_l,\ j\in\mathbb{N}\}.$$

NPEDF In this case $\mathcal{P}_{k,n}$ is included in all previous instances of τ_k, all instances of higher priority layers and instances of other tasks of the layer with deadline shorter or equal to $D_{k,n} = A_{k,n} + \overline{D}_{k,n}$. We say included, because of the specific priority order among instances with the same absolute deadline which we ignore by taking a conservative bound, see [8].

$$S_{\mathcal{P}_{k,n}}(U_{k,n}, t^+) \leqslant S_{k,0..n-1}(U_{k,n}, t^+) + \overline{S}_k(U_{k,n}, t^+, A_{k,n} + \overline{D}_k) + S_{1..\overline{m}_{l-1}}(U_{k,n}, t^+).$$

Given a family of MWAF's of the form $\mathcal{S} = \{\widehat{S}_i(x,d,q), \widehat{S}^P_k(x,d,q) \mid i = 1, \dots, \overline{m}_l,\ q \in Q\}$, a family of MWAF's as required by Lemma 4.1 consists in

$$\widetilde{S}^B_k(x,a,q) = \min\big(\widehat{S}_k(x^+,q), \widehat{S}^P_k(a^+,q)\big) + \widehat{\overline{S}}_k(x, a + \overline{D}_k, q) + \widehat{S}_{1..\overline{m}_{l-1}}(x,q)$$
$$\widetilde{S}^E_k(x,a,q) = \widehat{S}_k(a^+, a + \overline{D}_k, q) + \widehat{S}_{1..\overline{m}_{l-1}}(x,q) + \widehat{\overline{S}}_k(b, a + \overline{D}_k, q),$$

since $\mathcal{Q}_{k,n} = \{\tau_{i,j} \mid i \leqslant \overline{m}_{l-1}\}$. Because for all $\tau_{i,j} \in \lambda_l$, $\vec{Q}_{i,j} = (l,0,0)$ and $\vec{P}_{i,j} = (l, A_{i,j} + \overline{D}_{i,j}, i, j)$ the bound (4.1) on the blocking periods from lower priority tasks is

$$\widetilde{Z}_k = \max\{C_{i,j} \mid \tau_i \in \lambda_l,\ j \in \mathbb{N},\ \overline{D}_i \geqslant \overline{D}_k\}$$

NPLP In the case where the layers are scheduled non-preemptively, $\mathcal{Q}_{k,n} = \emptyset$. As a result, $\widetilde{S}^E_k(x,a,q)$ becomes independent of x, because once the execution has started, the instance can not be interrupted. Thus, $\widetilde{E}_k(a,q) = \widetilde{B}_k(a,q)$. For a task in a NPFP layer, this means

$$\widetilde{E}_k(a,q) = \widehat{S}_k(a^+,q) + \widehat{S}_{1..k-1}(\widetilde{B}_k(a,q),q) + \widetilde{Z}_k \tag{4.14}$$

and in a NPEDF layer

$$\widetilde{S}^E_k(x,a,q) = \widehat{S}_k(a^+, a + \overline{D}_k, q) + \widehat{\overline{S}}_k(\widetilde{B}_k(a,q), a + \overline{D}_k, q) + \widehat{S}_{1..\overline{m}_{l-1}}(\widetilde{B}_k(a,q),q) + \widetilde{Z}_k.$$

In both cases the MWAF is independent of x, implying that $\widetilde{E}_k(a,q) = \widetilde{B}_k(a,q) + C$, for some $C > 0$. On the other hand, the bound $\widetilde{Z}_k$ that accounts for the blocking periods due to lower priority tasks get larger, because $\tau_{k,n}$ can also be blocked by instances from lower priority layers. For NPFP it is

$$\widetilde{Z}_k = \max\{C_{i,j} \mid i \geqslant \overline{m}_l\ j \in \mathbb{N}\}$$

and

$$\widetilde{Z}_k = \max\{C_{i,j} \mid \tau_i \in \lambda_l,\ j \in \mathbb{N},\ \overline{D}_i \geqslant \overline{D}_k \text{ or } i > \overline{m}_l\}$$

for a NPEDF layer.

4.2 Response time bounds in the presence of non-preemptable resources

We intend to derive a generic response time bound for a task with time independent priority functions in a context of layered priorities, where the PCP is used to protect non-preemptive resources. We will use it in section ?? to derive response time bounds under different policies.

4.2.1 A generic Bound

The starting point is the execution end bound $\overline{E}_{k,n}$ given by (3.34) on page 30:

$$\overline{E}_{k,n} = \min\{t > U_{k,n} \mid \int_{U_{k,n}}^t \Pi_{\mathcal{B}_{k,n}}(x)dx + S_{\mathcal{P}_{k,n}}(U_{k,n},t) + U_{k,n} = t\}.$$

For the term $S_{\mathcal{P}_{k,n}}(U_{k,n}, t)$ we suppose that a set $\mathcal{P}_k = \{\widetilde{S}_k(x, a, q) \mid q \in Q\}$ of MWAF's is known such that $\forall\, \omega \in \Omega, \forall\, n \in \mathbb{N},\ \exists\, q \in Q$:

$$S_{\mathcal{P}_{k,n}}(U_{k,n}, t; \omega) \leqslant \widetilde{S}_k(t - U_{k,n}, A_{k,n} - U_{k,n}, q). \tag{4.15}$$

On the other hand we will derive a more specific bound, for $\int_{U_{k,n}}^{t} \Pi_{\mathcal{B}_{k,n}}(x)dx$. The definition of $\mathcal{B}_{k,n}$, see (3.33) on page 30, can be rewritten as

$$\mathcal{B}_{k,n} = \{\tau_{i,j} \mid \overline{\Gamma}_{i,j}(t) \prec \overline{\Gamma}_{k,n}(0)\ \forall\, t;\ \Gamma_{i,j}(U_{k,n}) \succ \overline{\Gamma}_{k,n}(0)\}.$$

According to (3.35) on page 30, there is only one instance of $\mathcal{B}_{k,n}$ which is pending in $[U_{k,n}, E_{k,n}[$. This instance is $\tau_{i,j}$. Since being part of $\mathcal{B}_{k,n}$ implies

$$\Gamma_{i,j}(U_{k,n}) \succ \overline{\Gamma}_{k,n}(0) \succ \overline{\Gamma}_{i,j}(U_{k,n}),$$

Proposition 3.10 applies for $[t_1, t_2[= [U_{k,n}, E_{k,n}[$. Thus there is a resource ζ_h used by $\tau_{i,j}$ at $U_{k,n}$, with $\int_{U_{k,n}}^{t} \Pi_{i,j}(x)dx \leqslant \widehat{Z}_i^h$ and $A_{i,j} < U_{k,n} \leqslant A_{k,n}$. Since $\tau_{i,j}$ is pending at $U_{k,n}$, $\vec{Q}^h(A_{i,j}) \succ \overline{\Gamma}_{k,n}(0)$. Thus the general bound is

$$\begin{aligned} \widetilde{Z}_k = \max\{\widehat{Z}_i^h \mid\ & \exists\, t < A_{k,n}\ \vdash\ \vec{Q}^h(t) \succ \overline{\Gamma}_{k,n}(0), \\ & \exists\, j, c\ \vdash\ Z_{i,j}^h(c) = 1,\ \overline{\Gamma}_{k,n}(0) \succ \overline{\Gamma}_{i,j}(t),\ \forall\, t\}. \end{aligned} \tag{4.16}$$

It accounts for critical sections of certain task on certain resources. If we define

$$\widetilde{E}_k(a, q) \stackrel{\mathsf{def}}{=} \min\{x > 0 \mid \tilde{Z}_k + \widetilde{S}_k(x, a, q) = x\} \qquad \mathcal{A}_k(q) \stackrel{\mathsf{def}}{=} \{a > 0 \mid \widetilde{S}_k(x, a, q) > x,\ \forall x \leqslant a\}, \tag{4.17}$$

the response time bound takes the same form as in Lemma 4.1.

Lemma 4.3 *Consider a task τ_k with time independent priority function that is part of a scheduling policy that that uses the PCP to protect non-preemptable resources. A bound on the response time of τ_k is*

$$R_{k,n} \leqslant \widetilde{R}_k \stackrel{\mathsf{def}}{=} \max_{q \in Q} \max_{a \in \mathcal{A}_k(q)} \widetilde{E}_k(a, q) - a,$$

where $\widetilde{E}_k(a, q)$ is given by (4.17).

The proof is similar to the proof of Lemma 4.1.

4.2.2 Time independent priority functions in layers

In the case of layered priorities, the set $\mathcal{P}_{k,n}$ on which the response time analysis is based (3.31), always contains all instances from higher priority layers. Notice that this fact is true whatever policies are used in the different layers. Thus, from the point of view of a task $\tau_{k,n}$ in some layer λ_l, all task from higher priority layers behave in the same way and all tasks from lower priority layers behave in some other common way.

In [8], response time bounds are derived for several time independent policies. The bounds are also based on characteristic functions $\widetilde{S}_k(x, a, q)$ which have to be adapted here to account for the preemption from tasks in higher priority layers. We suppose a family of MWAF's for the set of tasks is known.

FPP-layer: the formula of $\widetilde{S}_k(x, a, q)$ remains the same, but here the tasks $\tau_1, \ldots, \tau_{\overline{m}_{l-1}}$ are then part of higher priority layers:

$$\widetilde{S}_k(x, a, q) = \min\big(\widehat{S}_k(x, q), \widehat{S}_k(a^+, q)\big) + \widehat{S}_{1..k-1}(x, q).$$

EDF-layer: the preemption from higher priority layers is independent of the deadline of the task under study:

$$\widetilde{S}_k(x,a,q) = \widehat{S}_{1..\overline{m}_{l-1}}(x,q) + \widehat{S}_{\underline{m}_l..\overline{m}_l}(x, a+\overline{D}_k, q).$$

FIFO-layer: similarly to EDF:

$$\widetilde{S}_k(x,a,q) = \widehat{S}_{1..\overline{m}_{l-1}}(x,q) + \widehat{S}_{\underline{m}_l..\overline{m}_l}(a^+, q).$$

LIFO-layer: as for FPP, the form remains the same:

$$\widetilde{S}_k(x,a,q) = \widehat{S}_{1..\overline{m}_l}(x,q).$$

In all for cases, the bound $\widetilde{Z}_k^h$ is given by (4.16). In Section 5.4, we will give a numerical example.

4.2.3 Execution begin until execution end

Proposition 4.4 *Let τ_k be a task with PPEB priority assignment. Suppose a family of increasing functions $\mathcal{P}_k = \{\widehat{S}_k^Q(x,q) \mid q \in Q\}$ is known such that $\forall\, \omega \in \Omega$ and $\forall\, u \in \mathbb{R}_+$, $\exists\, q \in Q$:*

$$S_{\mathcal{Q}_{k,n}}(u, u+x) \leqslant \widehat{S}_k^Q(x,q) \quad \forall\, x > 0. \tag{4.18}$$

Let $\widehat{C}_k \geqslant C_{k,n}(\omega)$, $\forall\, \omega$. Then a bound for the time between execution beginning and end is

$$E_{k,n} - B_{k,n} \leqslant \max_{q\in Q}\ \min\{x > 0 \mid \widehat{C}_k + \widehat{S}_k^Q(x,q) = x\}. \tag{4.19}$$

Proof: Recall formula (3.36). Given Proposition A.4, the definition of $\widehat{C}_k$ and (4.18), the bound (4.19) follows immediately. ■

Notice that, there is only one partial bound per q, whereas for response times there may be several. The reason is that at the execution beginning of an instance, also the previous instance of the same task has completed.

4.3 Round Robin

In this section we are interested in the response times of an instance with a priority function of RR type (3.43), in the context of layered priorities with PCP. For this purpose we first have to derive some properties.

In order to determine a response time bound for a task, it is necessary to bound the preemption from other tasks. Under Round Robin, priorities change over time. No fixed set of higher priority instances exits. However, the function $P(.)$ allows to estimate the preemption. Let $\widehat{Z}_i = \max_h \widehat{Z}_i^h$ the longest critical section of τ_i.

Proposition 4.5 *A task $\tau_i \in \lambda_l$ in a Round Robin layer runs for at most Ψ_i units of time in each Round Robin cycle: if $W_{\lambda_l}(t) > 0$ for $\in]t_1, t_2[$ then*

$$\int_{t_1}^{t_2} \Pi_i(x)dx \leqslant \Psi_i \cdot \big(1 + P(t_2^-) - P(t_1)\big) + \widehat{Z}_i. \tag{4.20}$$

Proof: For the proof we subdivide the instances into several classes. Let τ_{i,j_1} be the first instance of τ_i to finish its execution strictly after t_1, i.e. so that $E_{i,j_1-1} \leqslant t_1 < E_{i,j_1}$. Similarly let τ_{i,j_2} be the last instance starting its execution strictly before t_2, i.e. so that $B_{i,j_2} < t_2 \leqslant B_{i,j_2+1}$. We suppose $j_1 \leqslant j_2$, otherwise there would be no instance of τ_i executed during $[t_1, t_2[$ and (4.20) would trivially be satisfied.

In order to establish the bound on $\int_{t_1}^{t_2} \Pi_i(x)dx$, it is convenient to partition the instances $\tau_{i,j}$, with $j_1 \leqslant j \leqslant j_2$, according to common reference points $A_{i,u_{i,j}}$ in the definition of $P_{i,j}(.)$:

$$e_0 = \min\{j \geqslant j_1 \mid P_{i,j}(A_{i,j+1}) < P(A_{i,j+1})\} \tag{4.21}$$

$$b_l = e_{l-1} + 1 \qquad l \geqslant 1 \tag{4.22}$$

$$e_l = \min\{j \geqslant b_l \mid P_{i,j}(A_{i,j+1}) < P(A_{i,j+1})\} \qquad l \geqslant 1. \tag{4.23}$$

Let $j_0 = \max\{b_l \leqslant j_2\}$. By Definition (4.23), for $\tau_{i,j}$ with $b_l < j \leqslant e_l$, $P_{i,j-1}(A_{i,j}) \geqslant P(A_{i,j})$. Furthermore, $b_l = u_{i,b_l} = u_{i,e_l} = u_{i,j}$.

For $t \in [E_{i,e_l}, B_{i,b_{l+1}}[$, $\Pi_i(t) = 0$, since $b_{l+1} = e_l + 1$. Thus, we can ignore the parts $\int_{E_{i,e_l}}^{B_{i,b_{l+1}}} \Pi_i(x)dx$. The remaining parts are the following:

- $\int_{t_1}^{E_{i,e_0}} \Pi_i(x)dx$:
  - $t_1 \leqslant A_{i,j_1}$: since $E_{i,j_1-1} < t_1$, $\Pi_i(x) = 0$ for $x \in [t_1, A_{i,j_1}[$. Thus, the integral reduces to $\int_{A_{i,j_1}}^{E_{i,e_0}} \Pi_i(x)dx$. It follows from (3.48), that $P_{i,j_1}(A_{i,j_1}) \geqslant P(A_{i,j_1})$. Using furthermore that $P(.)$ is monotonically increasing allows to prove the following inequality:

    $$\begin{aligned} P_{i,e_0}(E_{i,e_0}) - P(t_1) \geqslant P_{i,e_0}(E_{i,e_0}) - P(A_{i,j_1}) &\geqslant P_{i,e_0}(E_{i,e_0}) - P_{i,j_1}(A_{i,j_1-1}) \\ &= \lceil C_{i,u_{k,j_1}..e_0}/\Psi_i \rceil - \lceil C_{i,u_{k,j_1}..j_1-1}/\Psi_i \rceil \end{aligned}$$

    Now, since $\lceil x \rceil \leqslant x + 1$, we have $-\lceil x \rceil \geqslant -x - 1$. Thus

    $$\begin{aligned} \lceil C_{i,u_{k,j_1}..e_0}/\Psi_i \rceil - \lceil C_{i,u_{k,j_1}..j_1-1}/\Psi_i \rceil &\geqslant (C_{i,u_{k,j_1}..e_0} - C_{i,u_{k,j_1}..j_1-1})/\Psi_i - 1 \\ &= C_{i,j_1..e_0}/\Psi_i - 1 = (1/\Psi_i) \cdot \int_{A_{i,j_1}}^{E_{i,e_0}} \Pi_i(x)dx - 1 \end{aligned}$$

  - $A_{i,j_1} < t_1$: since $t_1 < Ei, j_1$, equation (3.48) implies $P_{i,j_1}(t_1) \geqslant P(t_1)$. Thus,

    $$P_{i,e_0}(E_{i,e_0}) - P(t_1) \geqslant P_{i,e_0}(E_{i,e_0}) - P_{i,j_1}(t_1) = (1/\Psi_i) \cdot \left\lceil \int_{t_1}^{E_{i,e_0}} \Pi_i(x)dx \right\rceil \geqslant \int_{t_1}^{E_{i,e_0}} \Pi_i(x)dx/\Psi_i$$

- $\int_{B_{i,b_l}}^{E_{i,e_l}} \Pi_i(x)dx$: since $u_{i,e_l} = u_{i,b_l} = b_l$, we have

  $$P_{i,e_l}(E_{i,e_l}) - P_{i,b_l}(B_{i,b_l}) = \lceil C_{i,b_l..e_l}/\Psi_i \rceil \geqslant C_{i,b_l..e_l}/\Psi_i = (1/\Psi_i) \cdot \int_{B_{i,b_l}}^{E_{i,e_l}} \Pi_i(x)dx.$$

- $\int_{B_{i,j_0}}^{t_2} \Pi_i(x)dx$: let $t_0 = \max\{t \leqslant t_2 \mid \Gamma_{i,j_2}(t) = \overline{\Gamma}_{i,j_2}(t)\}$. For $[t_0, t_2[$ we use Proposition 3.10:

  $$\int_{t_0}^{t_1} \Pi_i(x)dx \leqslant \widehat{Z}_i^h \leqslant \widehat{Z}_i,$$

  for some ζ_h, used by τ_i. By the definition of t_0, $P_{i,j_2}(t_0) = P(t_0)$. The function $P(.^-)$ is increasing on $]t_1, t_2]$. The left-hand side limit is needed, because when $W_{\lambda_l}(t_2) = 0$ then $P(t_2) = 0$, recall the definition. Thus,

  $$\begin{aligned} P(t_2^-) - P_{i,j_0}(B_{i,j_0}) &\geqslant P(t_0) - P_{i,j_2}(B_{i,j_0}) \\ &\geqslant P_{i,j_0}(t_0) - P_{i,j_0}(B_{i,j_0}) = \lceil \textstyle\int_{B_{i,j_0}}^{t_0} \Pi_i(x)dx/\Psi_i \rceil \geqslant (1/\Psi_i) \cdot \int_{B_{i,j_0}}^{t_0} \Pi_i(x)dx, \end{aligned}$$

where we have also used that, by definition of $j_0 = u_{i,j_0}$. Thus,

$$\begin{aligned}\int_{t_1}^{t_2} \Pi_i(x)dx &= \int_{t_1}^{E_{i,e_0}} \Pi_i(x)dx + \sum_{l\geqslant 1}^{e_l=j_2-1} \int_{B_{i,b_l}}^{E_{i,e_l}} \Pi_i(x)dx + \int_{B_{i,j_2}}^{t_0} \Pi_i(x)dx + \int_{t_0}^{t_1} \Pi_i(x)dx \\ &\leqslant \Psi_i \cdot \left(1 + P_{i,e_0}(E_{i,e_0}) - P(t_1) + \sum_{l\geqslant 1}^{e_l=j_2-1} \left(P_{i,e_l}(E_{i,e_l}) - P_{i,b_l}(B_{i,b_l})\right) + P(t_2) - P_{i,j_0}(B_{i,j_0})\right) + Z_i \\ &\leqslant \Psi_i \cdot (1 + P(t_2) - P(t_1)).\end{aligned}$$

The last line is obtained using that

$$P_{i,e_{l-1}}(E_{i,e_{l-1}}) = P_{i,e_{l-1}}(A_{i,b_l}) < P(A_{i,b_l}) \leqslant P_{i,b_l}(A_{i,b_l}) = P_{i,b_l}(B_{i,b_l}).$$

■

At $A_{k,u_{k,n}}$, the workload of the task τ_k is zero, but the workload of task from higher priority layers may be positive, which is impractical for deriving response time bounds. The last time before $A_{k,u_{k,n}}$, where their workload is zero is preferable. Accordingly, let $U_{k,n}$ be such that

$$U_{k,n} \leqslant A_{k,u_{k,n}}, W_{1..\overline{m}_{l-1}}(U_{k,n}) = 0 \text{ and } W_{1..\overline{m}_{l-1}}(t) > 0 \quad t \in]U_{k,n}, A_{k,u_{k,n}}]. \tag{4.24}$$

Notice that $W_k(t) = 0$ for $t \in [U_{k,n}, A_{k,u_{k,n}}]$.

For the derivation of response time bounds of $\tau_{k,n}$, we have to analyze the set of higher priority instances $\mathcal{H}_{k,n}(t)$. Let λ_l be the Round Robin layer that contains $\tau_{k,n}$. The set $\mathcal{H}_{k,n}(t)$ contains all instances from higher priority layers λ_s, $s < l$, but also certain instances form lower priority layers, that use non-preemptable resources with a priority ceiling in the RR layer or higher. Let $\tau_{i,j} \in \lambda_s$, $s > l$. Then

$$\Gamma_{i,j}(U_{k,n}) \succ \Gamma_{k,n}(U_{k,n}) \quad \Leftrightarrow \quad \Gamma_{i,j}(U_{k,n}) \succcurlyeq (l).$$

To handle that kind of preemption we redefine $\mathcal{B}_{k,n}$ in this context by

$$\mathcal{B}_{k,n} = \{\tau_{i,j} \in \lambda_s \mid s > l,\ \Gamma_{i,j}(U_{k,n}) \succcurlyeq (l)\}.$$

From Proposition 3.10 it is known that at $U_{k,n}$ at most one instance of $\mathcal{B}_{k,n}$ is pending and executing a critical section before $B_{k,n}$. Accordingly, let

$$\widetilde{Z}_k = \max\{\widehat{Z}_i^h \mid \tau_i \in \lambda_s,\ s > l,\ \exists\, j, c\ \vdash\ Z_{i,j}^h(c) = 1\}.$$

It is also convenient to introduce a notation for the sum of the scheduling functions of other tasks in the RR layer λ_l, to which the instance under study belongs:

$$\overline{\Pi}_k(x) = \sum_{\tau_i \in \lambda_l, i \neq k} \Pi_i(x).$$

Lemma 4.6 *The workload of higher priority instances is strictly positive between the beginning of the interference period defined by* (4.24) *and the activation time:*

$$W_{\mathcal{H}_{k,n}(t)}(t) > 0 \qquad \forall\, t \in [U_{k,n}, A_{k,n}[. \tag{4.25}$$

Proof: We divide the interval into two parts: $[U_{k,n}, A_{k,n}[= [U_{k,n}, A_{k,u_{k,n}}] \cup]A_{k,u_{k,n}}, A_{k,n}[$. The definition of $U_{k,n}$ implies that $W_{1..\overline{m}_{l-1}}(t) > 0$ for $t \in [U_{k,n}, A_{k,u_{k,n}}]$. Since $\tau_{i,j} \in \mathcal{H}_{k,n}(t)$, if $i \leqslant \overline{m}_{l-1}$, we have $0 < W_{1..\overline{m}_{l-1}}(t) < W_{\mathcal{H}_{k,n}(t)}(t)$.

We turn now to $t \in]A_{k,u_{k,n}}, A_{k,n}[$. Let $\tau_{k,s}$ such that $u_{k,n} < s \leqslant n$. For $t \in]A_{k,s-1}, E_{k,s-1}[$, we have $W_{k,s-1}(t) > 0$, and because $\Gamma_{k,s-1}(t) \succ \Gamma_{k,n}(t)$, we have $\tau_{k,s-1} \in \mathcal{H}_{k,n}(t)$. Thus, (4.25) holds on this interval.

For $t \in [E_{k,s-1}, A_{k,s}]$, we have to distinguish two cases:

- $P_{k,s-1}(A_{k,s}) > P(A_{k,s})$: since $P_{k,s-1}(.)$ is constant after $E_{k,s-1}$, we have $P_{k,s-1}(t) > P(A_{k,s}) \geqslant P(t)$. Thus, there must be a pending instance $\tau_{i,j} \in \lambda_l$ with, $P(t) \leqslant P_{i,j}(t) < P_{k,s-1}(t)$. Since $\Gamma_{i,j}(t) \succ \Gamma_{k,s-1}(t) \succ \Gamma_{k,n}(t)$, $\tau_{i,j} \in \mathcal{H}_{k,n}(t)$ and (4.25) holds.
- $P_{k,s-1}(A_{k,s}) = P(A_{k,s})$, $k < K(A_{k,s})$: since $P_{k,s-1}(.)$ is constant after $E_{k,s-1}$, we have $P_{k,s-1}(t) = P(A_{k,s}) = P(t)$. Since $k < K(A_{k,n})$, there must be an instance $\tau_{i,j} \in \lambda_l$ with $K(A_{k,n}) \geqslant i > k$. Again, $\tau_{i,j} \in \mathcal{H}_{k,n}(t)$ and (4.25) holds.

■

Theorem 4.7
The execution end of an instance in a Round Robin layer is bounded by

$$\overline{E}_{k,n} = \min\{t > U_{k,n} \mid C_{k,u_{k,n}..n} + \int_{U_{k,n}}^{t} \Pi_{\mathcal{B}_{k,n}}(x)dx + S_{1..\overline{m}_{l-1}}(U_{k,n},t) + \int_{U_{k,n}}^{t} \overline{\Pi}_k(x)dx + U_{k,n} = t\}. \tag{4.26}$$

Proof: Recall the scheduling function based execution end formula (2.40) on page 17.

$$E_{k,n} = \min\{t > A_{k,n} \mid C_{k,n} + \int_{A_{k,n}}^{t} \Pi_{\mathcal{H}_{k,n}(x)}(x)dx + A_{k,n} = t\}. \tag{2.40}$$

It is an equation of the form (A.2), with $x_0 = A_{k,n}$ and $f(t) = C_{k,n} + \int_{A_{k,n}}^{t} \Pi_{\mathcal{H}_{k,n}(x)}(x)dx$, for $t > A_{k,n}$.

First, we extend the domain of definition by $[U_{k,n}, A_{k,n}[$, with the help of Proposition A.3. Let

$$g(t) = C_{k,n} + \int_{U_{k,n}}^{t} \Pi_{\mathcal{H}_{k,n}(x)}(x)dx.$$

Property (4.25), implies $\Pi_{\mathcal{H}_{k,n}(x)}(x) = 1$, for $t \in [U_{k,n}, A_{k,n}[$. Thus, $\int_{U_{k,n}}^{A_{k,n}} \Pi_{\mathcal{H}_{k,n}(x)}(x)dx = A_{k,n} - U_{k,n}$ and therefore $f(t) + A_{k,n} = g(t) + U_{k,n}$, i.e. condition (A.3) holds. Furthermore, $g(t) = C_{k,n} + t - U_{k,n} > t - U_{k,n}$, meaning that also (A.4) is satisfied. So far we have proven

$$E_{k,n} = \min\{t > U_{k,n} \mid C_{k,n} + \int_{U_{k,n}}^{t} \Pi_{\mathcal{H}_{k,n}(x)}(x)dx + U_{k,n} = t\}.$$

The set of higher priority instances $\mathcal{H}_{k,n}(x)$ varies during time, due to the RR scheduling in the layer and the PCP. However, $\mathcal{H}_{k,n}(x)$ is contained in the set of instances that either belong to tasks of higher priority layers, to other tasks of the layer, to τ_k or are part of $\mathcal{B}_{k,n}$:

$$\mathcal{H}_{k,n}(x) \subset \{\tau_{i,j} \mid i \leqslant \overline{m}_{l-1};\ \text{or}\ \tau_{i,j} \in \lambda_l,\ i \neq k;\ \text{or}\ i = k,\ j < n;\ \text{or}\ \tau_{i,j} \in \mathcal{B}_{k,n}\}.$$

The sum of the scheduling functions of these tasks are a bound:

$$\begin{aligned} g(t) =& C_{k,n} + \int_{U_{k,n}}^{t} \Pi_{\mathcal{H}_{k,n}(x)}(x)dx \\ \leqslant& \int_{U_{k,n}}^{t} \left(\Pi_{1..\overline{m}_{l-1}}(x) + \overline{\Pi}_k(x) + \Pi_{k,0..n-1}(x) + \Pi_{\mathcal{B}_{k,n}}(x)\right)dx \\ \leqslant& C_{k,n} + S_{1..\overline{m}_{l-1}}(U_{k,n},t) - W_{1..\overline{m}_{l-1}}(t) + S_{k,0..n-1}(U_{k,n},t) - W_{k,0..n-1}(t) \\ &+ \int_{U_{k,n}}^{t} \left(\overline{\Pi}_k(x) + \Pi_{\mathcal{B}_{k,n}}(x)\right)dx \\ \leqslant& S_{1..\overline{m}_{l-1}}(U_{k,n},t) + C_{k,0..n} + \int_{U_{k,n}}^{t} \left(\overline{\Pi}_k(x) + \Pi_{\mathcal{B}_{k,n}}(x)\right)dx \stackrel{\text{def}}{=} \hat{g}(t). \end{aligned} \tag{4.27}$$

Proposition A.5 applies to $g(.)$ and $\hat{g}(.)$, with $x_0 = \hat{x}_0 = U_{k,n}$, which directly implies (4.26). ■

Notice that instead of $S_{k,u_{k,n}..n}(U_{k,n},t)$ we could have used $C_{k,u_{k,n}..n}$. But the equation would then be based on a larger function, since $S_{k,u_{k,n}..n}(U_{k,n},t) \leqslant C_{k,u_{k,n}..n}$, for $t > U_{k,n}$. It does not make any difference when considering the execution end of some instance $\tau_{k,n}$ with the corresponding interference period beginning $U_{k,n}$, but it is of some importance for deriving response time bounds from a family of MWAF's. The aim of using $S_{k,u_{k,n}..n}(U_{k,n},t)$ is to avoid conservative bounds.

Proposition 4.8 *Let λ_l be a layer scheduled according to the Round Robin policy. Suppose a family of MWAF's $\mathcal{S}$ is known for the task set (Definition 2.9). A bound on the response times of a task $\tau_k \in \lambda_l$ is*

$$\max_{q\in Q} \max_{\tau_{i,j}\in\mathcal{A}_k(q)} \widetilde{E}_{k,j}(q) - \hat{A}_{k,j}(q), \tag{4.28}$$

where

$$\widetilde{E}_{k,j}(q) = \min\{x > 0 \mid \min(\hat{f}^1_{k,j}(x,q), \hat{f}^2_{k,j}(x,q)) = x\}, \tag{4.29}$$

$$\hat{f}^1_{k,j}(x,q) = \widetilde{Z}_k + \widehat{\overline{Z}}_k + \widehat{C}_{k,0..j}(q) + \lceil \widehat{C}_{k,0..j}(q)/\Psi_k \rceil \cdot \overline{\Psi}_k + \widehat{S}_{1..\overline{m}_{l-1}}(x,q), \tag{4.30}$$

$$\begin{aligned}\widetilde{Z}_k = \max\{\widehat{Z}^h_i \mid \exists\, t < A_{k,n} \ \vdash\ \vec{Q}^h(t) \succ \overline{\Gamma}_{k,n}(0),\\ \exists\, j, c \ \vdash\ Z^h_{i,j}(c) = 1,\ i > \overline{m}_l\}\end{aligned}$$

$$\hat{f}^2_{k,j}(x) = \widetilde{Z}_k + \max_u \{\widehat{\overline{S}}_k(0, u+x) + \widehat{S}_{1..\overline{m}_{l-1}}(0, u+x) + \widehat{S}_k(0,u) - u\} \tag{4.31}$$

and $\mathcal{A}_k(q) = \{\tau_{k,j} \mid \hat{f}^2_{k,j}(x) + \widehat{S}_k(x,q) > x, \forall\, x \leqslant \hat{A}_{k,j}(q)\}$.

Proof: The execution end formula (4.26) is in the form (A.2), with $x_0 = U_{k,n}$ and

$$f(t) = C_{k,u_{k,n}..n}(q) + \textstyle\int_{U_{k,n}}^t \Pi_{\mathcal{B}_{k,n}}(x)dx + S_{1..\overline{m}_{l-1}}(U_{k,n}, t) + \int_{U_{k,n}}^t \overline{\Pi}_k(x)dx.$$

We intend to apply Proposition A.5. For this purpose we will derive bounds for the different terms that appear in $f(t)$. For the considered trajectory ω let q be the corresponding element from the family of MWAF's for the task set. Notice that for non-deadline based MWAF's, corresponding sequences of activations $\hat{A}_{k,j}(q)$ with $\widehat{C}_{k,j}(q)$ do always exist.

- Let $\tilde{n} \stackrel{\text{def}}{=} \max\{j \mid \hat{A}_{k,j}(q) \leqslant A_{k,n} - U_{k,n}\}$. We have

$$C_{k,u_{k,n}..n} = S_k(U_{k,n}, A^+_{k,n}) \leqslant \widehat{S}_k(0, (A_{k,n} - U_{k,n})^+, q) = \widehat{C}_{k,0..\tilde{n}}(q).$$

- Notice first that $x \in [U_{k,n}, A_{k,u_{k,n}}[$, $\overline{\Pi}_k(x) = 0$. We have then, using Proposition 4.5:

$$\begin{aligned}\textstyle\int_{U_{k,n}}^t \overline{\Pi}_k(x)dx &= \textstyle\int_{A_{k,u_{k,n}}}^t \overline{\Pi}_k(x)dx \leqslant \int_{A_{k,u_{k,n}}}^{E_{k,n}} \overline{\Pi}_k(x)dx \leqslant \big(P(E^-_{k,n}) - P(A_{k,u_{k,n}}) + 1\big) \cdot \Psi_k\\ &= \lceil C_{k,u_{k,n}..n}/\Psi_k \rceil \cdot \Psi_k \leqslant \lceil \widehat{C}_{k,0..j}(q)/\Psi_k \rceil \cdot \Psi_k\end{aligned}$$

- $\int_{U_{k,n}}^t \Pi_{\mathcal{B}_{k,n}}(x)dx \leqslant \widetilde{Z}_k$.

- $S_{1..\overline{m}_{l-1}}(U_{k,n}, t) \leqslant \widehat{S}_{1..\overline{m}_{l-1}}(t - U_{k,n}, q)$.

This leads to the bound $\hat{f}^1_{k,j}(.)$ On the other hand

$$\textstyle\int_{U_{k,n}}^t \overline{\Pi}_k(x)dx = \overline{S}_k(U_{k,n}, t) + \overline{W}_k(U_{k,n}) = \overline{S}_k(U_{k,n}, t) + \overline{S}_k(U^m_{k,n}, U_{k,n}) - \int_{U^m_{k,n}}^{U_{k,n}} \Pi_k(x)dx$$

and using $1 = \Pi_k(t) + \overline{\Pi}_k(t) + \Pi_{\mathcal{B}_{k,n}}(t) + \Pi_{1..\overline{m}_{l-1}}(t)$,

$$= \overline{S}_k(U^m_{k,n}, t) - (U_{k,n} - U^m_{k,n}) + \textstyle\int_{U^m_{k,n}}^{U_{k,n}} \Pi_{\mathcal{B}_{k,n}}(x)dx + \int_{U^m_{k,n}}^{U_{k,n}} \Pi_k(x)dx + \int_{U^m_{k,n}}^{U_{k,n}} \Pi_{1..\overline{m}_{l-1}}(x)dx$$

and that $W_{1..\overline{m}_{l-1}}(U^m_{k,n}) = W_{1..\overline{m}_{l-1}}(U_{k,n}) = 0$ and $W_k(U^m_{k,n}) = W_k(U_{k,n}) = 0$,

$$= \overline{S}_k(U^m_{k,n}, t) - (U_{k,n} - U^m_{k,n}) + \int_{U^m_{k,n}}^{U_{k,n}} \Pi_{\mathcal{B}_{k,n}}(x)dx + S_k(U^m_{k,n}, U_{k,n}) + S_{1..\overline{m}_{l-1}}(U^m_{k,n}, U_{k,n}).$$

Thus,

$$\begin{aligned}
&\int_{U_{k,n}}^t \overline{\Pi}_k(x)dx + \int_{U_{k,n}}^t \Pi_{\mathcal{B}_{k,n}}(x)dx + S_{1..\overline{m}_{l-1}}(U_{k,n}, t)\\
&\qquad = \overline{S}_k(U^m_{k,n}, t) + \int_{U^m_{k,n}}^t \Pi_{\mathcal{B}_{k,n}}(x)dx + S_{1..\overline{m}_{l-1}}(U^m_{k,n}, t) + S_k(U^m_{k,n}, U_{k,n}) - (U_{k,n} - U^m_{k,n})\\
&\qquad \leqslant \hat{f}^2_{k,j}(t - U_{k,n}),
\end{aligned}$$

where $\hat{f}^2_{k,j}(.)$ is given by (4.31).

Now, with $\hat{f}(x) = \min(\hat{f}^1_{k,j}(x), \hat{f}^2_{k,j}(x))$, we obtain, by Proposition 4.5 $E_{k,n} - U_{k,n} \leqslant \widetilde{E}_{k,\tilde{n}}$. On the other hand $\hat{A}_{k,\tilde{n}} \leqslant A_{k,n} - U_{k,n}$. Thus

$$R_{k,n} = E_{k,n} - A_{k,n} = E_{k,n} - U_{k,n} - (A_{k,n} - U_{k,n}) \leqslant \widetilde{E}_{k,\tilde{n}} - \hat{A}_{k,\tilde{n}}.$$

The remaining question is if $\hat{A}_{k,\tilde{n}} \in \mathcal{A}_k(q)$. Since $W_{\mathcal{H}_{k,n}(x)}(x) > 0$ for $t \in [U_{k,n}, A_{k,n}[$,

$$\begin{aligned}
0 &< g(t) - (t - U_{k,n})\\
&\leqslant S_{1..\overline{m}_{l-1}}(U_{k,n}, t) + S_{k,0..n-1}(U_{k,n}, t) + \int_{U_{k,n}}^t \left(\overline{\Pi}_k(x) + \Pi_{\mathcal{B}_{k,n}}(x)\right)dx\\
&\leqslant S_{1..\overline{m}_{l-1}}(U_{k,n}, t) + S_k(U_{k,n}, t) + \overline{S}_k(U_{k,n}, t) + \widetilde{Z}_k\\
&\leqslant \hat{f}^2_{k,j}(t, q) + S_k(U_{k,n}, t).
\end{aligned}$$

■

5 Numerical applications

5.1 Layered priorities

The efficiency of EDF, concerning feasibility is due to the fact that if the execution time of a task is increased, the resulting additional preemption is fairly distributed over all tasks. Under EDF the response times increase less for tasks with short deadlines than for tasks with large deadlines. Consider now the task set given in Table 1.a. Suppose that the characteristics of τ_9 are difficult to estimate and that instead of the assumed maximum $C_9 = 80$ it may actually need $C_9 = 121$ units of time. As can be seen in Table 1.b, the task set is infeasible. But more interesting, all tasks just fail to meet their deadlines, which is the *domino effect* (see[11]).

Suppose the tasks $\tau_1, \ldots, \tau_7$ have hard deadlines, whereas $\tau_8, \ldots, \tau_{10}$ have soft deadlines, which reflect more the relative importance than a deadline to be met. In this case, a solution could consist in moving the tasks with hard deadlines in a higher priority layer. As can be seen in Table 1.c, the tasks $\tau_1, \ldots, \tau_7$ are protected, when $C_9 = 121$.

Notice that with this solution it is not necessary to supervise the actual execution times of $\tau_8, \ldots, \tau_{10}$, to guarantee that the tasks with hard deadlines are feasible. This may be an interesting advantage although the layers reduce the ability of EDF to produce a feasible schedule. Furthermore, compared to FPP (see Table1.d), EDF reduces the maximal response time of the task $\tau_8, \ldots, \tau_{10}$.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
edf	t10	40	800	700	637	63	5.0	90.8
	t9	**80**	1000	600	537	63	8.0	85.8
	t8	35	250	250	187	63	14.0	77.8
	t7	20	400	200	137	63	5.0	63.8
	t6	25	150	150	87	63	16.7	58.8
	t5	6	150	120	57	63	4.0	42.1
	t4	2	40	20	15	5	5.0	38.1
	t3	8	70	15	13	2	11.4	33.1
	t2	3	20	10	8	2	15.0	21.7
	t1	2	30	5	3	2	6.7	6.7

(a) Task set under EDF.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
edf	t10	40	800	700	701	-1	5.0	94.9
	t9	**121**	1000	600	601	-1	12.1	89.9
	t8	35	250	250	251	-1	14.0	77.8
	t7	20	400	200	201	-1	5.0	63.8
	t6	25	150	150	151	-1	16.7	58.8
	t5	6	150	120	121	-1	4.0	42.1
	t4	2	40	20	21	-1	5.0	38.1
	t3	8	70	15	16	-1	11.4	33.1
	t2	3	20	10	11	-1	15.0	21.7
	t1	2	30	5	6	-1	6.7	6.7

(b) Task set under EDF at the limit of schedulability.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
edf	t10	40	800	700	782	-82	5.0	94.9
	t9	**121**	1000	600	682	-82	12.1	89.9
	t8	35	250	250	332	-82	14.0	77.8
edf	t7	20	400	200	96	104	5.0	63.8
	t6	25	150	150	56	94	16.7	58.8
	t5	6	150	120	26	94	4.0	42.1
	t4	2	40	20	15	5	5.0	38.1
	t3	8	70	15	13	2	11.4	33.1
	t2	3	20	10	8	2	15.0	21.7
	t1	2	30	5	3	2	6.7	6.7

(c) Tasks protected in a higher priority layer.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
fpp	t10	40	800	700	892	-192	5.0	94.9
	t9	**121**	1000	600	688	-88	12.1	89.9
	t8	35	250	250	195	55	14.0	77.8
	t7	20	400	200	96	104	5.0	63.8
	t6	25	150	150	56	94	16.7	58.8
	t5	6	150	120	24	96	4.0	42.1
	t4	2	40	20	15	5	5.0	38.1
	t3	8	70	15	13	2	11.4	33.1
	t2	3	20	10	5	5	15.0	21.7
	t1	2	30	5	2	3	6.7	6.7

(d) Same task set under FPP.

Table 1: EDF and layers.

5.2 Round Robin

To give an idea of the behavior of the Round Robin scheduling policy, we will consider the sample task set shown in Table 2. It is not possible to schedule the entire task set under Round Robin, unless choosing some Ψ_k smaller than 1 to keep the maximal length of the RR cycles small enough. If $\Psi_k = 1$ $\forall\, k$, then $\Psi^l = 13$. But τ_1 requires then two RR cycles to complete, and τ_3 even three. They may miss their deadlines, because at the release time of an instance the RR scheduler may just have checked the corresponding task so that the instance must wait an entire cycle before being reconsidered. Feasibility may achieved with $\Psi_k < 1$ but this seems excessively small compared to the execution time $C_{13} = 100$. For an instance of τ_{13} this could mean more than 100 interruptions.

On the other hand, a layer consisting in the tasks $\tau_6, \ldots, \tau_{10}$ can be scheduled under Round Robin. The result is shown in Table 3. The column entitled p, gives possible assignment POSIX priorities (a higher numerical value means also a higher priority in that context). To compare the two alternatives, we have performed simulation and estimated the mean and maximal response times and its standard deviation.

It appears that when the layer is scheduled under Round Robin, the maximum of the average response time jitters in the layer is smaller under Round Robin than under FPP. The response time jitter is measured by its standard deviation. Under FPP we have obtained 52.8 whereas only 34.7 under RR.

To understand this comparison consider the following. It can be verified that the jitter of the response time of a task tends to be lower when its priority is higher. Thus, if its average response time jitter of a task is higher than acceptable, one would increase its priority. But other tasks may then become infeasible. We conclude therefore that being able to reduce the jitter without using higher priorities is interesting if small average response time jitter is a matter of concern.

In Table 3, we have chosen as POSIX priority $p = 4$, to underline that being part of the RR layers is as being scheduled at lowest priority of the range $4, \ldots, 8$, for the FPP layer in Table 2. In the example, the maximum of the average response time jitters of $\tau_6, \ldots, \tau_{10}$ remains close to 50, even if the priorities $4, \ldots, 8$ are differently attributed to these tasks. On the other hand, if they are scheduled under Round Robin as in Table 3, then the maximum drops to 34.7, and this without using a higher priority.

Layer	p	τ_k	C_k	T_k	D_k	bound	max	mean	stdev	ρ_k	$\rho_{1..m}$
	13	T1	2	40	10	2	2.0	2.0	0.0	5.0	5.0
	12	T2	3	20	10	5	5.0	3.5	0.7	15.0	20.0
	11	T3	1	30	15	6	6.0	2.3	2.1	3.3	23.3
	10	T4	7	70	20	13	13.0	9.2	2.1	10.0	33.3
	9	T5	6	150	30	19	19.0	9.2	3.7	4.0	37.3
fpp	8	T6	11	300	190	34	33.0	24.8	4.8	3.7	41.0
	7	T7	35	250	230	90	90.0	59.0	10.5	14.0	55.0
	6	T8	20	400	230	114	114.0	50.0	24.6	5.0	60.0
	5	T9	25	400	400	167	166.0	100.0	31.0	6.2	66.3
	4	T10	40	700	700	227	226.0	122.1	**52.8**	5.7	72.0
	3	T11	40	800	800	367	338.0	197.4	52.6	5.0	77.0
	2	T12	80	1000	1100	630	624.0	376.2	130.2	8.0	85.0
	1	T13	100	1400	1400	1392	1387.0	730.3	267.0	7.1	92.1

Table 2: A task set under FPP only ...

Layer	p	Ψ_k	τ_k	C_k	T_k	D_k	bound	max	mean	stdev	ρ_k	$\rho_{1..m}$
fpp	13		T1	2	40	10	2	2.0	2.0	0.0	5.0	5.0
	12		T2	3	20	10	5	5.0	3.5	0.7	15.0	20.0
	11		T3	1	30	15	6	6.0	2.3	2.1	3.3	23.3
	10		T4	7	70	20	13	13.0	9.2	2.1	10.0	33.3
	9		T5	6	150	30	19	19.0	9.2	3.7	4.0	37.3
Round Robin	4	4	T6	11	300	190	180	162.0	48.7	25.7	3.7	41.0
	4	9	T7	35	250	230	227	216.0	77.1	30.4	14.0	55.0
	4	5	T8	20	400	230	227	208.0	93.0	32.6	5.0	60.0
	4	7	T9	25	400	400	227	195.0	90.6	29.3	6.2	66.3
	4	10	T10	40	700	700	227	200.0	100.0	**34.7**	5.7	72.0
fpp	3		T11	40	800	800	367	338.0	197.4	52.6	5.0	77.0
	2		T12	80	1000	900	630	624.0	376.2	130.2	8.0	85.0
	1		T13	100	1400	1400	1392	1387.0	730.3	267.0	7.1	92.1

Table 3: ... and with a Round Robin layer.

5.3 Non-preemptive policies in preemptive layers

It is interesting to notice that priority functions directly allow to define non-preemptive policies in preemptive layers. In that case, an instance of the layer may be preempted by a instance of a higher priority layer but not by an instance of its own layer. Such a policy could be useful in the following situation. Suppose a set of task such as shown in Table 4 should be scheduled non-preemptively for some reason. As can be seen, the set of tasks consists in two kinds of tasks, whose parameters differ by an order of magnitude. Under NP-EDF, the "small" tasks τ_1 and τ_2 are not feasible because of the blocking from the "larger" tasks, which appears when a small task is just activated after a "large" task has started to execute. A solution could consist in grouping the "small" tasks in a higher priority layer and the "larger" task in a lower priority layer and to schedule the layers preemptively. As can be seen, the set is feasible, because the "small" tasks are allowed to preempt the "large" tasks. With this policy some of the non-preemptive behavior is kept. If for example, the "large" tasks need a non-preemptive resource, which is not needed by the "small" tasks, then no semaphore is required.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
np-edf	task5	10	70	50	42	8	14.3	80.4
	task4	15	100	70	59	11	15.0	66.1
	task3	12	80	100	60	40	15.0	51.1
	task2	1	9	10	**17**	-7	11.1	36.1
	task1	2	8	7	**16**	-9	25.0	25.0

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
np-edf	task5	10	70	50	42	8	14.3	80.4
	task4	15	100	70	59	11	15.0	66.1
	task3	12	80	100	60	40	15.0	51.1
np-edf	task2	1	9	10	**3**	7	11.1	36.1
	task1	2	8	7	**2**	5	25.0	25.0

Table 4: Non-preemptive policies in preemptive layers.

5.4 Bounds on blocking periods under PCP

Table 5 shows response time bounds for a set of tasks scheduled under preemptive layered priorities with priority ceiling protocol for non-preemptive resources. Critical sections induce bounded periods of priority inversions where a higher priority task is temporally preempted by a lower priority task. In this section we will illustrate the bounds $\widetilde{Z}_k$ (4.16) on page 49 on these blocking periods.

There are two resources ζ_1 and ζ_2. The highest priority instances that may use ζ_1 belong to τ_2, which is scheduled in a FPP layer, λ_1. Thus, according to (3.39) on page 33, the associated ceiling is

$$\vec{Q}^1(t) = (1, 2).$$

The highest priority instances that may use ζ_2 belong to tasks which are scheduled in the EDF layer λ_2. Thus, according to (3.40) on page 34, the priority ceiling is based on the shortest relative deadline of the tasks that may use ζ_2, which is $\overline{D}_7 = 250$. Therefore,

$$\vec{Q}^2(t) = (2, t + 250).$$

As example we derive $\widetilde{Z}_3$ according to (4.16) on page 49. Since $\vec{Q}^1(t) = (1,2) \succ (1,3,n)\ \forall\ t \in \mathbb{R}_+,\ n \in \mathbb{N}$, the ceiling associated with ζ_1 can induce blocking periods for τ_3. On the other hand $\vec{Q}^2(t) = (2, t+250) \prec (1,3,n)$ and thus the ceiling associated with ζ_2 is to be ignored. It remains to determine which instances $\tau_{i,j}$ may block an instance $\tau_{3,n}$, by having its priority promoted to $\vec{Q}^1(A_{i,j})$. These are the instances of τ_5 and τ_7, but not τ_3, since the latter always have a higher priority than $\tau_{3,n}$. Since $\widehat{Z}^1_5 = 2$ and $\widehat{Z}^1_7 = 10$, we conclude that the bound on the length of a blocking period is $\widetilde{Z}_3 = 10$.

Layer	Task	C^{max}	T^{min}	D^{min}	Z^{max}	bound	laxity	ρ_k	$\rho_{1..m}$
Round Robin	t11	20	800	800	Z^2=9	615	185	2.5	86.6
	t10	50	1000	1000	Z^1=2	655	345	5.0	84.1
	t9	70	900	900		655	245	7.8	79.1
edf	t8	20	400	400	Z^2=7	137	263	5.0	71.3
	t7	35	250	250	Z^2=3, Z^1=10	111	139	14.0	66.3
	t6	40	800	800	Z^2=5	219	581	5.0	52.3
	t5	6	150	150	Z^1=2	53	97	4.0	47.3
fpp	t4	7	70	70		30	40	10.0	43.3
	t3	2	40	40		18	22	5.0	33.3
	t2	5	20	20	Z^1=3	16	4	25.0	28.3
	t1	1	30	30		1	29	3.3	3.3

Table 5: Effects of critical sections under the priority ceiling protocol

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
Round Robin	t11	20	800	800	592	208	2.5	86.6
	t10	50	1000	1000	639	361	5.0	84.1
	t9	70	900	900	639	261	7.8	79.1
edf	t8	20	400	400	115	285	5.0	71.3
	t7	35	250	250	89	161	14.0	66.3
	t6	40	800	800	195	605	5.0	52.3
	t5	6	150	150	26	124	4.0	47.3
fpp	t4	7	70	70	15	55	10.0	43.3
	t3	2	40	40	8	32	5.0	33.3
	t2	5	20	20	6	14	25.0	28.3
	t1	1	30	30	1	29	3.3	3.3

Table 6: Same task set but without critical sections

6 Conclusion

In this document, we have illustrated the the concept of priority functions introduced in [8] by defining the class of layered priority based scheduling policies and we have shown how to realize the Round Robin scheduling policy in terms of priority functions. Furthermore, we have given a generic form of the priority ceiling protocol, which can be applied to any preemptive policy. In this context the deadlock problem with nested critical sections has given some insight into the meaning of the proof of existence of scheduled task processes, which must be established for any policy defined in the framework of the trajectory based model. The proof of existence ensures in the particular case of preemptable resource that no deadlock occurs when the priority ceiling protocol is employed.

An other interesting point to notice is the recurrent use of left or right-continuity in the definition of the different concepts. If two events have a cause and effect relation, one usually considers that the earlier event influences the later one. Represented in a discrete event system model, these events may appear simultaneously, because a model is kept as less complex as possible. In order to obtain an accurate model, cause and effect relations must be represented in some way by the model. In our trajectory based model, it is the left or right-continuity of functions defined on $\mathbb{R}_+$ that specifies how simultaneous events influence each other.

We have derived response time bounds for tasks scheduled under layered priorities, Round Robin and the priority ceiling protocol, to allow feasibility testing. It should be noticed that we do not have assumed any particular task model. The response time bounds are valid for any task model for which a family of MWAF's is known. This approach has the advantage of allowing a separate study of tasks and scheduling policies, which reduces the difficulty of the analysis.

The basic motivation of this study is to enlarge the class of scheduling policies that can be used for real-time systems, by providing the analysis that allows feasibility testing. The aim is to enable a refined choice, which does not only guarantee feasibility, but does also provide additional properties that improve the quality of a particular system. We have seen for example that Round Robin allows to reduce (average) response time jitter. Which properties the different policies can provide is an open question that need further investigation.

A Miscellaneous properties

In this section we give some technical properties, which are useful for deriving response time bounds. We omit the proofs; they can by found in [8].

A.1 Fixed point equations and iterative computation

A.1.1 Fixed point

Response times are often expressed as particular solutions of fixed point equations. These equations generally have more than one fixed point and such that in their neighborhood, the involved function is not necessarily a contraction. This partially explains why response times appear as "first fixed point after some point", see (A.1). In this section we give several simple properties of these fixed point equations which are repeatedly used when deriving response time equations.

Lemma A.1 *Let f be an increasing function $\mathbb{R} \longmapsto \mathbb{R}$ with $f(0) > 0$. Define*

$$\mathcal{X} \stackrel{\mathsf{def}}{=} \{x > 0 \mid f(x) = x\} \qquad\qquad x^* \stackrel{\mathsf{def}}{=} \inf \mathcal{X} \quad (\textit{if } \mathcal{X} \neq \emptyset).$$

We have the following properties:

1. *If there exists an $\hat{x} > 0$ such that $f(\hat{x}) \leqslant \hat{x}$ then there exists a $\tilde{x} \leqslant \hat{x}$ such that $f(\tilde{x}) = \tilde{x}$.*
2. *If $\mathcal{X} \neq \emptyset$ then $x^* = \min \mathcal{X}$,*
3. *and $f(x) > x$ for $x \in [0, x^*[$.*

The response time $R_{k,n} = x^*$ of a task typically satisfies an equation of the form

$$x^* = \min\{x > 0 \mid f(x_0 + x) = x \} \tag{A.1}$$

with $x_0 = A_{k,n}$ and f being the appropriate work arrival function. The related execution end $E_{k,n} = A_{k,n} + R_{k,n}$ is also solution of a fixed point equation. On can pass from one to the other, by changing the dummy variable $x = y - x_0$, which only affects the way of writing, but not the involved function:

$$y^* = x_0 + x^* = \min\{y > x_0 \mid f(y) + x_0 = y \}. \tag{A.2}$$

Lemma A.2 *Equation* (A.1) *is equivalent to* (A.2).

Notice that the point $x_0 = A_{k,n}$ plays the role of a reference epoch after which something is considered. Furthermore the involved function f does not need to be defined before x_0. But if the execution end is expressed using the beginning of the appropriate interference period $U_{k,n}$, then the reference point changes to $y_0 = U_{k,n}$. The following proposition helps to handle this situation.

Proposition A.3 *Let $y_0 < x_0$ and g be an increasing function $\mathbb{R} \longmapsto \mathbb{R}$ which coincides with f up to an additive constant such that*

$$g(y) = f(y) + x_0 - y_0 \quad \forall\, y > x_0. \tag{A.3}$$

If

$$g(y) > y - y_0 \;\; \forall\, y \in [y_0, x_0[, \tag{A.4}$$

then for y^* defined in (A.2),

$$y^* = \min\{y > y_0 \mid g(y) + y_0 = y \}.$$

If f can be bounded by another function $\hat{f}$ then the first fixed point either remains unchanged or is shifted towards the future $x^* \leqslant \hat{x}^*$. This is typically used when deriving response times bounds with $x_0 = U_{k,n}$ being the beginning of the interference period containing the instance under study and $\widehat{x}_0 = 0$ being the beginning of the first interference period of the majorizing task process.

Proposition A.4 *Let $\hat{f}$ be an increasing function $\mathbb{R} \longmapsto \mathbb{R}$ with $\hat{f}(\hat{x}_0) > \hat{x}_0$ for some $\hat{x}_0 \in \mathbb{R}$, and such that*

$$\hat{x}^* = \min\{x > 0 \mid \hat{f}(\hat{x}_0 + x) = x\ \} \tag{A.5}$$

exists. If

$$\forall\, x \in [0, \hat{x}^*[\quad f(x_0 + x) \leqslant \hat{f}(\hat{x}_0 + x), \tag{A.6}$$

then x^* exists and $x^* \leqslant \hat{x}^*$. If furthermore

$$\widehat{f}(x^*) = x^*, \tag{A.7}$$

then $\hat{x}^* = x^*$.

Proposition A.5 *Let $\hat{f}$ be an increasing function $\mathbb{R} \longmapsto \mathbb{R}$ with $\hat{f}(\hat{x}_0) > 0$ for some $\hat{x}_0 \in \mathbb{R}$, and such that*

$$\hat{y}^* = \min\{y > \hat{x}_0 \mid \hat{f}(y) + \hat{x}_0 = y\ \} \tag{A.8}$$

exists. If

$$\forall\, x \in [0, \hat{x}^*[\quad f(x_0 + x) \leqslant \hat{f}(\hat{x}_0 + x), \tag{A.9}$$

then $\hat{y}^*$ exists and $y^* - x_0 \leqslant \hat{y}^* - \hat{x}_0$. If furthermore

$$\hat{f}(y^* - x_0 + \hat{x}_0) + x_0 = y^*, \tag{A.10}$$

then $\hat{y}^* - \hat{x}_0 = y^* - x_0$.

Corollary A.6
If $f(x_0 + x) \leqslant \sigma + \rho \cdot x$, $\forall x$, for some $\sigma, \rho \in \mathbb{R}_+$ with $\rho < 1$ then $x^* = \min\{x > 0 \mid f(x) = x\}$ exists.

A.1.2 Iterations

The iterative computation of response times has been introduced by Joseph and Pandya [6]. Here we state this method in the framework of our model and give the assumption under which the computation works.

First remind the following definition. A function $f : \mathbb{R} \longmapsto \mathbb{R}$ is said to be *piecewise constant* if on any interval of finite length there is a finite partition into sub-intervals where the function is constant.

Proposition A.7 *Let $f : \mathbb{R}_+ \longmapsto \mathbb{R}_+$ be a left-continuous and increasing function with $f(0) > 0$, such that $x^* = \min\{x > 0 \mid f(x) = x\}$ exists. Let $x_0 \in [0, x^*[$ and $x_n = f(x_{n-1})$, for $n \leqslant 1$.*

(i) The sequence (x_n) converges to x^*.

(ii) If f is piecewise constant, then the convergence takes a finite number of steps.

Notice that assuming f to take integer values would have the same effect than assuming it to be piecewise constant. In view of the application of the model, discrete time and thus discrete WAF could be justified and furthermore it would not change the feasibility problem [1], but since such an assumption is not necessary we keep the model as general as possible.

List of Notations

$\prec$, **12**
$\preccurlyeq$, **12**
(A, C, D), **5**
(A, C, D, Π), **7**
$(\mathcal{P}, \preccurlyeq)$, **11**
(a, c, d), **5**
(a, c, d, π), **7**

$\mathcal{A}_k(q)$, 43, 53
$A_{k,n}$, **5**, 9
$A_{k,n,s}$, **26**

$\mathcal{B}_{k,n}$, **20**, 51
$\widetilde{B}_k(a, q)$, 43
$B_{k,n}$, **8**
$B_{k,n,s}$, **27**

$C_{k,n}$, **5**, 9
$C_{k,n,0..s}$, **25**
$C_{k,n1..n_2}$, **35**
$C_{k,n,s}$, **25**

$\overline{D}_{k,n}$, **5**, 9
$D_{k,n}$, 5, **9**

$\widetilde{E}_k(a, q)$, 43
$E_{k,n}$, **8**
$E_{k,n,s}$, **26**

$\overline{\Gamma}_{k,n}(t)$, **25**
$\Gamma_{k,n}(t)$, **11**
$\Gamma_{k,n,s}(t)$, **26**

$\mathcal{H}_{k,n}(t)$, **15**

$K(t)$, **34**

λ_l, **32**
$\mathcal{L}_{k,n}(t)$, 15

m, **5**
$\mathcal{M}_{k,n}$, **21**
$\underline{m}_l$, **32**
$\overline{m}_l$, **32**

ν, **7**

Ω, **5**
ω, **5**

$\Pi_k(t, d)$, **8**
$\Pi_{k,n}(t, d)$, **8**
$\overline{\Pi}_k(x)$, 51
$\mathcal{P}$, **11**
$\mathcal{P}_{k,n}$, **20**
$\vec{P}_{k,n}$, **18**
$\vec{P}_{k,n,s}$, **26**
$\Pi_{1..m}(t)$, **10**
$\Pi_{k,n}(t)$, **7**
$P_{k,n}(t)$, **35**
$P(t)$, **34**

$\mathcal{Q}_{k,n}$, **20**
$\vec{Q}^h(t)$, **25**, **32**
$\vec{Q}_{k,n}$, **18**
$\vec{Q}_{k,n,s}$, **26**

$\rho_{1..m}$, **10**
ρ_k, **10**
R_k^{max}, **9**
$R_{k,n}$, **8**

$\sigma_{1..m}$, **10**
σ_k, **10**
$\mathbb{S}$, **11**
$\widetilde{S}_k^B(x, a, q)$, 43
$\widetilde{S}_k^E(x, a, q)$, 43
$\widetilde{S}_k(x, a, q)$, 48
$S_{1..m}(t_1, t_2)$, **10**
$S_{\mathcal{I}}(t_1, t_2)$, **6**
$S_k(t_1, t_2)$, **6**
$S_k(t_1, t_2, d)$, **6**
$S_{k,n}(t_1^+, t_2)$, **6**
$S_{k,n}(t_1, t_2)$, **6**
$S_{k,n}(t_1, t_2^+)$, **6**
$S_{k,n}(t_1, t_2, d)$, **8**
$S_k^P(t_1, t_2)$, 44

τ_k, **5**
$\tau_{k,n}$, **5**
$\mathbb{T}$, **5**
$\mathbb{T}_0$, **10**
$\mathbb{T}_0^\Pi$, **10**
$\mathbb{T}^\Pi$, **7**
$\mathcal{T}$, **5**
$T_{k,n}$, **5**, 9

$U_{k,n}$, **20**, **51**

$u_{k,n}$, **35**

$W_{1..m}(t)$, **10**
$W_k(t,d)$, **8**
$W_{k,n}(t)$, **8**, 9
$W_{k,n}(t,d)$, **8**
$W_{k,n}(x^+)$, 8

Ψ_k, **33**
Ψ_k^l, **34**

ζ_h, **24**
$\widehat{Z}_i$, **49**
$\widetilde{Z}_k$, 43, 48, 51
$\widehat{Z}_k^h$, **23**
$Z_{k,n}^h(c)$, **24**
$Z_{k,n}^h(c^+)$, **24**

Index

absolute deadline, 5, **9**
activation, **5**
  time, **5**

CAN, **38**
completion time, 8
controller area network, *see* CAN
cycle time, **5**

deadline
  absolute, 5, **9**
  relative, **9**
deadlock, 30, 37
discrete time, 61

earliest deadline first, *see* EDF
EDF, 14, 32
equivalent priority assignments, **16**
execution
  beginning, **8**
  end, **8**
  time, **5**

feasibility, **9**
FIFO, 14, 32
first in first out, *see* FIFO
fixed point equation, **60**
fixed preemptive priorities, *see* FPP
FPP, 14, 32

highest priority first, *see* HPF
history, **5**
HPF, **12**

instance, **5**
  pending, **8**
  sub-, **25**
inter arrival time, **5**
interference period
  priority promotion at B, 20
invocation, **5**
iterative computation, 61

last in first out, *see* LIFO
layered preemptive priorities, *see* LPP
LIFO, 14, 32
LPP, **32**

mark, **5**

non-idling, **10**
non-preemptable resource, **24**, 37
non-preemptive earliest deadline first, *see* NP-EDF
non-preemptive fixed priorities, *see* NP-FP
non-preemptive last in first out, *see* NP-LIFO
non-preemptive policy, **38**
NP-EDF, 39
NP-FP, 39
NP-LIFO, 39

PCP, 24
PCP, dynamic, 24
piecewise constant, 61
point, **5**
  of accumulation, **5**
  process, **5**
policy
  non-preemptive, **38**
priorities
  layered preemptive, *see* LPP
priority, **12**
  assignment, **11**
    decidable, **12**
    EDF, 14
    equivalent, **16**
    FIFO, 14
    FPP, 14
    LIFO, 14
    NP-EDF, 39
    NP-FP, 39
    NP-LIFO, 39
    piecewise order preserving, **12**
    PPEB, 18
    Round Robin, 33
  ceiling
    principle, **25**
    protocol, *see* PCP
  promotion, *see* PPEB, PCP
priority inversion, 18
priority promotion at execution beginning, *see* PPEB
properly nested resources, **23**

recurrent task, **5**
relative deadline, **9**
release time, **5**

resource, **24**
  non-preemptable, **24**, 37
  properly nested, **23**
response time, **8**
Round Robin, *see* RR
RR, 11, **33**

scheduling function, **7**
scheduling policy
  deterministic, **7**
  random, **7**
(σ, ρ)-bound, **10**
stable, **10**
step-function, 6
sub-instance, **25**

task
  recurrent, **5**
task process, **5**
  scheduled, **7**
  stable, **10**
task sequence, **5**
task set, **5**
time
  activation, **5**
  completion, **8**
  cycle, **5**
  execution, **5**
  inter arrival, **5**
  release, **5**
  response, **8**
trajectory, **5**

WAF, **6**
work arrival function, **6**
  deadline based, **6**
  majorizing, **11**
    family of, **11**
workload function, 8

References

[1] S.K. Baruah, L.E. Rosier, and R.R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2, 1990.

[2] M. Chen and K. Lin. Dynamic priority ceiling: A concurrency control protocol for real-time systems. *RTS*, 2, 1990.

[3] R.L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):132–142, January 1991.

[4] R. Davis. Dual priority scheduling: A means of providing flexibility in hard real-time systems. Technical Report YCS230, Univerity of York, May 1994.

[5] (ISO/IEC). *9945-1:1996 (ISO/IEC)[IEEE/ANSI Std 1003.1 1996 Edition] Information Technology - Portable Operating System Interface (POSIX) - Part 1 : System Application : Program Interface*. IEEE Standards Press, 1996. ISBN 1-55937-573-6.

[6] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[7] I.D. Katcher, H. Arakawa, and J.K. Strosinder. Engineereing and analysis of fixed priority schedulers. *IEEE-T-SE*, 19(9):920–934, September 1993.

[8] J.M. Migge and A. Jean-Marie. Timing analysis of real-time scheduling policies: A trajectory based model. Research Report 3561, INRIA, November 1998. `ftp://ftp.inria.fr/INRIA/publication/RR/RR-3561.ps.gz`.

[9] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[10] M. Spuri. Analysis of deadline scheduled real-time systems. Technical Report 2772, INRIA, January 1996.

[11] J. Stankovic, M. Spuri, K. Ramamritham, and C. Buttazzo. *Deadline Scheduling for Real-Time Systems*. Kluwer Academic Publishers, 1998.

[12] K. Tindell, A. Burns, and A.J. Wellings. An extendible approach for analysing fixed priority hard real time sytems. *Real-Time Systems*, 6(2), 1994.

[13] K. Tindell, A. Burns, and A.J. Wellings. Calculating controler area network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399