Thèse de Doctorat en Informatique

Conservatoire National des Arts et Métiers

présentée par

Franck BIMBARD

le 23 novembre 2007

pour obtenir

Le titre de docteur en informatique

Dimensionnement temporel de systèmes embarqués : application à OSEK

Jury

Directeur de Thèse :	Eric GRESSIER-SOUDAN	CNAM, Paris
Co-Directeur :	Laurent GEORGE	LISSI, Université Paris 12
Rapporteurs :	Guy JUANOLE	LAAS, Toulouse
	Zoubir MAMMERI	IRIT, Toulouse
Examinateurs :	Joël GOOSSENS	ULB, Bruxelles, Belgique
	Pierre LEBEE	Société VirtualLogix
	Yves SOREL	Projet AOSTE INRIA Rocquencourt
	Claude VILLARD	INT, Evry

Remerciements

Avant tout, j'adresse mes sincères remerciements à Eric GRESSIER-SOUDAN et Claude VILLARD pour leur soutien tout au long de ces trois années de thèse. Je ne saurais dire combien nos échanges et leurs nombreux conseils m'ont été précieux.

Je tiens à exprimer toute ma gratitude à Laurent GEORGE qui m'a accueilli dans son équipe et m'a permis de mener à bien ce travail. Nos discussions, toujours très fructueuses, ont beaucoup compté dans l'orientation de mes recherches et l'aboutissement de ces trois années d'études.

Je remercie vivement Guy JUANOLE et Zoubir MAMMERI d'avoir accepté d'être les rapporteurs de ma thèse. Leurs remarques et conseils, tous très constructifs, m'ont beaucoup aidé.

Je suis très reconnaissant à Joël GOOSSENS, Pierre LEBEE et Yves SOREL d'avoir participé à mon jury de thèse et les en remercie sincèrement.

Je souhaite également remercier mes anciens collègues et amis Céline BARTH, Chrystelle GUEGAN, Parinaz LAHMI, Jean-Marc BAUDON et Sio-Hoï IENG pour leur sympathie et leur bonne humeur.

Enfin, je remercie mes collègues de l'École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise, et tout particulièrement Catherine DUBOIS et Olivier PONS, qui, en plus d'être toujours disponibles, m'ont constamment prodigué d'excellents conseils.

Bien sûr, je ne peux terminer sans remercier mes proches de tout coeur et notamment mes parents qui, au cours de ces trois années de thèse, m'ont toujours soutenu et encouragé, comme d'habitude...

Résumé

Cette thèse traite du dimensionnement temps réel de systèmes embarqués. Nous proposons un ensemble d'outils algorithmiques permettant de garantir, avant son déploiement, qu'une application, une fois installée sur une architecture monoprocesseur donnée, sera exécutée en temps réel. Nous nous plaçons dans un contexte temps réel strict avec des échéances de terminaison au plus tard. De plus, nous ne considérons que des applications constituées de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes et non concrètes. Le standard OSEK, étudié dans cette thèse, est basé sur un ordonnancement FP/FIFO et prescrit le mécanisme du plafond de priorité pour protéger les ressources. Cette étude commence naturellement par l'identification et la caractérisation des charges dues à notre exécutif OSEK. Puis, nous proposons des conditions de faisabilité, intégrant les charges précédentes, valables pour tout ensemble de tâches ordonnancées FP/FIFO et se partageant, au plus, une ressource. Bien que le standard OSEK n'admette que des priorités fixes, nous montrons comment mettre en oeuvre un ordonnancement EDF pour des tâches n'utilisant, cette fois, aucune ressource. Là encore, de nouvelles conditions de faisabilité, intégrant les charges cumulées du système d'exploitation et de notre implémentation, sont présentées. Enfin, nous expérimentons les conditions de faisabilité précédentes sur une plateforme réelle. Les résultats confirment que les charges dues au système d'exploitation ne peuvent être négligées. Ces expérimentations montrent également que nos conditions de faisabilité s'avèrent opérationnelles pour le dimensionnement temps réel d'applications embarquées.

Mots clés : OSEK, dimensionnement temps réel, conditions de faisabilité, ordonnancement FP/FIFO, ordonnancement EDF, temps de réponse pire cas, charges dues au système d'exploitation

Abstract

In this thesis, we are interested in real time dimensioning of embedded systems. We propose a set of algorithmic tools which allows developers to verify that their application will respect its real time constraints accordingly to a given monoprocessor architecture. We work in hard real time context with termination deadlines. In addition, we only consider periodic, preemptive or non-preemptive, independent and non-concrete tasks with arbitrary deadlines. The OSEK standard has been initiated in 1993 by several german companies. This standard is based on a FP/FIFO scheduling policy and protects each resource by using priority ceiling protocol. First of all we identify and measure the overheads of an OSEK kernel. We propose feasibility conditions taking previous overheads into account. These feasibility conditions can be used with tasks scheduled accordingly to FP/FIFO policy and using at most one resource. Although OSEK standard only accepts fixed priorities, we show how to implement EDF scheduling policy for tasks using no resource. Once again, we propose feasibility conditions taking into account the overheads due to the kernel and our implementation. Finally, our previous feasibility conditions are experimented on a real platform. These experimentations confirm that kernel overheads can not be neglected. It is also shown that our feasibility conditions are valid for real time dimensioning.

Keywords : OSEK, real time dimensioning, feasibility conditions, FP/FIFO, EDF, worst case response time, kernel overheads

Table des matières

Table des figures

Chapitre 1

Introduction

Cette thèse s'intéresse au dimensionnement temps réel de systèmes embarqués monoprocesseurs. Notre objectif consiste à mettre au point un outil pour garantir, avant son déploiement, qu'une application embarquée respectera toujours ses contraintes temporelles lorsque celle-ci sera exécutée sur une architecture monoprocesseur donnée. Force est de constater que, de nos jours, la technologie évolue en permanence et de plus en plus vite. A la section 1.a, nous évoquons le fait que cette évolution incessante répond à la demande du grand public mais, dans le même temps, l'alimente. A l'évidence, l'intégration de nouvelles fonctions, toujours plus perfectionnées, au sein de nos outils quotidiens, rend leur mise au point de plus en plus complexe, en particulier au niveau temps réel. Ce problème est discuté à la sous section 1.b qui souligne également l'intérêt du dimensionnement temps réel durant la conception d'un nouveau produit. Le standard OSEK, incontournable dans l'industrie automobile, est introduit à la section 1.c. Cette section justifie notre choix de développer notre outil de dimensionnement temps réel sur ce standard. Enfin, l'organisation du présent rapport est détaillée à la section 1.d.

1.a Introduction

De nombreux produits, financièrement inaccessibles au plus grand nombre dans les années 1990, se retrouvent aujourd'hui présents dans la plupart des foyers, voire entre les mains de tous. L'ordinateur personnel figure parmi les exemples les plus évidents de cette progression. Cette démocratisation est rendue possible par l'évolution permanente des technologies et la diminution de leur coût. Les nouvelles technologies s'accompagnant toujours de performances plus grandes permettent la conception de produits novateurs qui suscitent l'engouement du grand public ainsi qu'une demande croissante de sa part. Ce cercle sans fin justifie la durée de vie de plus en plus courte des produits actuels et se retrouve dans de nombreux domaines : l'automobile avec l'arrivée de l'assistance au parking, la correction d'assiette et la personnalisation de l'habitacle, l'aéronautique avec l'Airbus A380 qui regorge de nouvelles technologiques, ou encore la téléphonie mobile où de nouveaux services apparaissent fréquemment.

Obéissant à une intégration de plus en plus forte, la conception de ces nouveaux produits devient également de plus en plus complexe. Tout logiciel étant constitué de fonctions, également appelées tâches, le problème revient à vérifier que celles-ci seront toujours exécutées en temps et en heure. Or, dès lors qu'une nouvelle fonctionnalité est intégrée à un produit, celle-ci s'accompagne de nouvelles tâches qui affectent le comportement de celles déjà présentes dans son application et remet donc en question leur validation. Prenons pour exemple une automobile dotée, entre autres choses, d'un système ABS. A chaque ajout d'un nouveau dispositif d'assistance électronique, le constructeur se doit de vérifier que celui-ci fonctionnera correctement et n'entraînera aucune défaillance des systèmes déjà présents dans le véhicule. En conséquence, le constructeur doit s'assurer que l'ensemble des tâches, y compris les anciennes comme celles liées à la fonction ABS, respecteront toujours leur échéance.

Finalement, la complexité croissante des produits actuels nécessite de nouveaux outils afin que les concepteurs puissent valider correctement leurs applications. Les outils algorithmiques, développés tout au long de cette thèse, permettent de déterminer précisément, pour chaque tâche d'une application donnée, le temps maximum au bout duquel celle-ci aura terminé son exécution à partir du moment où son activation est survenue. Par exemple, grâce à ces outils, le concepteur d'une application automobile sait au bout de combien de temps, dès lors que le blocage d'une roue a été constaté, l'exécution de la fonction ABS sera assurément terminée. Ces outils sont d'autant plus précis qu'ils tiennent compte de l'exécution du système d'exploitation. Par ailleurs, comme nous l'avons dit précédemment, cette problématique n'est pas propre à l'industrie automobile et se retrouve dans de nombreux autres domaines.

1.b Position du problème

A la section précédente, le temps tient clairement le premier rôle. Ainsi, valider un système consiste à démontrer que l'exécution de n'importe laquelle de ses tâches se termine toujours avant l'échéance, aussi appelée contrainte temporelle, associée à celle-ci. Dans ce cas, le système est dit "temps réel". Attention toutefois à ne pas conclure qu'un système "temps réel" se doit forcément d'être rapide : dans le cas d'un traitement vidéo l'ensemble de l'image doit être traité en un temps inférieur à la persistance rétinienne, alors que l'industrie chimique peut autoriser une attente de quelques minutes pour arrêter une réaction. Ainsi, les contraintes temporelles dépendent du domaine applicatif et n'obligent pas toujours à déployer l'application sur un processeur des plus rapides. Cette thèse s'intéresse donc au problème de la garantie du respect des contraintes temporelles associées aux tâches qui constituent l'application. Nous soulignons que les outils algorithmiques, développés durant cette thèse, ne portent que sur le dimensionnement temps réel monoprocesseur et font l'hypothèse que l'algorithme et le code de chaque tâche sont corrects.

Le dimensionnement d'un système temps réel vise à valider, avant son déploiement, que n'importe laquelle de ses tâches respectera toujours sa contrainte temporelle. L'écart entre la date d'activation d'une tâche et celle à laquelle l'exécution correspondante se termine s'appelle le temps de réponse. Aucun retard par rapport aux échéances n'étant toléré, nous parlons de temps réel à contraintes strictes. Par opposition, le temps réel à contraintes relatives autorise le dépassement occasionnel des échéances. Avant tout, le dimensionnement d'un système requière sa modélisation. Puis, afin de garantir un dimensionnement valable pour tout scénario possible, nous optons pour une approche pire cas. Nous déterminons donc, pour chaque tâche, le scénario menant à son temps de réponse maximum appelé temps de réponse pire cas. Si le temps de réponse pire cas, de n'importe laquelle de ses tâches, est inférieur ou égal à son échéance, alors le système est validé. Nous précisons que l'algorithme d'ordonnancement, qui détermine l'ordre dans lequel les tâches s'exécutent, joue un rôle fondamental dans le calcul des temps de réponse pires cas.

A l'heure actuelle, les développeurs ne tiennent pas compte de la charge due au système d'exploitation. Traditionnellement, lors de la conception d'une application, ces derniers n'utilisent pas pleinement le processeur afin que le système d'exploitation puisse s'exécuter. Malheureusement, cette marge de sécurité, difficile à estimer correctement, s'avère la plupart du temps soit insuffisante, soit excessive. Dans les deux cas, les conséquences économiques s'avèrent non négligeables. En cas de sous-estimation de la charge due au système d'exploitation, les contraintes temporelles ne seront pas respectées. Selon les disfonctionnements occasionnés, un rappel des produits vendus peut alors devenir nécessaire. A l'inverse, en cas de surestimation de cette charge, l'impact économique tient principalement au fait que les capacités du processeur sont alors surdimensionnées par rapport aux besoins réels. Les outils algorithmiques, développés tout au long de cette thèse, tenant compte des charges dues au système d'exploitation, permettront de valider une application tout en tirant pleinement profit du processeur. Leur impact sur le plan économique apparaît évidemment positif.

1.c Application au standard du secteur automobile : OSEK

OSEK est l'abréviation de "Offene Systeme und deren Schnittstellen für die Electronik im Kraftfahrzeug" ce qui se traduit, en Français, par "systèmes ouverts et leurs interfaces pour l'électronique automobile". Créé en 1993, OSEK est le fruit d'une collaboration entre plusieurs industriels allemands dont BMW, Bosch, DaimlerChrysler, Opel, Siemens, et Volkswagen, ainsi qu'un département de l'Université de Karlsruhe. L'objectif de ce projet consistait à réduire les problèmes de compatibilité en proposant une interface standard pour tous les processeurs du marché ainsi que leurs interfaces de communication. En 1994, les constructeurs français Renault et PSA, qui développaient un projet similaire, rejoignirent le consortium OSEK. Ce standard s'impose, aujourd'hui, comme une référence parmi les différents systèmes d'exploitation temps réel disponibles pour les applications embarquées.

Nécessitant peu de ressources, le standard OSEK peut être implémenté sur un simple processeur 8 bits doté de quelques kilo-octets de mémoires vive et morte ou sur un processeur 32 bits doté de plusieurs centaines de kilo-octets de mémoires [49] [51]. Pour cette raison, ce standard est mis en oeuvre avec un grand nombre de processeurs de différentes marques telles que STMicroelectronics, Infineon, ou bien encore Microchip, sans oublier Motorola qui détient la plus grande part du marché automobile. Ainsi, ce standard améliore nettement la portabilité et l'interopérabilité des applications qui l'exploitent.

De plus, les applications du standard OSEK, et par conséquent l'intérêt de cette thèse, ne se limitent absolument pas au seul secteur automobile. En effet, de part leur robustesse et leur simplicité d'utilisation, beaucoup de technologies automobiles, telles que les bus CAN et LIN, se retrouvent employées dans de nombreux autres domaines comme le ferroviaire ou l'électroménager. C'est en premier lieu pour diminuer la longueur des faisceaux électriques et, par conséquent, les quantités de cuivre dans leurs véhicules que, depuis plusieurs années, les constructeurs automobiles utilisent, de plus en plus couramment, des réseaux multiplexés comme les bus CAN [13] et LIN [18]. En outre, son support bifilaire différentiel et sa correction d'erreurs automatique confèrent au bus CAN une grande robustesse. Le bus LIN, quant à lui, s'avère moins robuste que le bus CAN mais plus économique. Ces réseaux permettent d'interconnecter les différents processeurs présents au sein d'un véhicule. L'interconnection de plusieurs processeurs, équipés d'un système d'exploitation OSEK, grâce à un réseau CAN aboutit à un système "temps réel distribué" [34]. De plus, la mise en place de ce type de réseau est d'autant plus simple que la plupart des systèmes d'exploitation OSEK intègrent une couche de communication réseau [48]. Nous rappelons que l'étude menée durant cette thèse concerne le dimensionnement temps réel de systèmes monoprocesseurs et que le passage aux systèmes temps réel distribués demeure une perspective de ce travail.

1.d Organisation du rapport

Les techniques d'analyse classiques, c'est à dire ne tenant pas compte de la charge due au système d'exploitation, utilisées pour le dimensionnement temps réel sont présentées au chapitre 2. Dans un premier temps, ce chapitre rappelle les différents modèles de tâches, de contraintes temporelles et d'ordonnancement ainsi que les concepts et les notations usuels. Les principaux algorithmes d'ordonnancement sont ensuite étudiés. Conformément à notre approche pire cas, les scénarios menant aux temps de réponse pires cas sont détaillés. Enfin, ce chapitre expose les conditions de faisabilité classiques reposant sur le calcul et l'analyse de ces temps de réponse pires cas.

Le standard OSEK est ensuite résumé au chapitre 3. Ses différentes politiques d'ordonnancement ainsi que sa gestion des tâches y sont expliquées. Nous y verrons également comment la méthode du plafond de priorité, prescrite par le standard OSEK pour gérer l'accès aux ressources, élimine tout risque d'interblocage entre différentes tâches. Le mécanisme des alarmes, sur lequel nous nous basons pour rendre les tâches périodiques, y est également présenté. Enfin, les différentes charges, dues à notre noyau OSEK fourni par la société Vector et conforme à la spécification 2.2 de ce standard, y sont identifiées et caractérisées.

Le chapitre 4 présente notre analyse temps réel pour un système monoprocesseur basé sur une implémentation du standard OSEK. Cette analyse s'applique à tout ensemble de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes, non concrètes et se partageant, au plus, une ressource. Notre objectif consistant à calculer précisément le temps de réponse pire cas de chaque tâche, nous présentons, dans ce chapitre, nos conditions de faisabilité améliorées par la prise en compte des charges du système d'exploitation. Nous expliquons aussi comment considérer les tâches sporadiques et les interruptions, et comparons notre étude à d'autres travaux de recherche visant à perfectionner les conditions de faisabilité classiques.

Nous étudions, au chapitre 5, la politique d'ordonnancement EDF, basée sur des priorités dynamiques, qui s'avère plus optimale que la politique FP/FIFO, prescrite par le standard OSEK et basée sur des priorités fixes. Nous expliquons comment implémenter cet algorithme en nous appuyant sur les seules tâches à priorité fixe autorisées par le standard OSEK. Les calculs des temps de réponse pires cas pour des tâches préemptives et non préemptives, ordonnancées selon cette politique, y sont ensuite étudiés. Cette analyse est applicable à tout ensemble de tâches à échéance arbitraire, périodiques, indépendantes, non concrètes et n'utilisant aucune ressource.

Enfin, les conditions de faisabilité, mises au point au chapitre 4, pour un ordonnancement FP/FIFO, ont été confrontées à des mesures effectuées sur une plateforme réelle. De même, la politique d'ordonnancement EDF a été déployée sur notre noyau OSEK et ses conditions de faisabilité ont également été évaluées à l'aide de mesures réalisées dans un cas réel. Les résultats de ces expérimentations sont exposés au chapitre 6. Ces derniers montrent que la charge due au système d'exploitation ne peut être négligée et que nos conditions de faisabilité s'avèrent utiles au dimensionnement temps réel d'applications basées sur le standard OSEK.

Chapitre 2

Etat de l'art : Analyse temps réel

Par définition le dimensionnement d'un système consiste à établir l'ensemble de ses dimensions. Dans notre cas, un système équivaut à un ensemble de tâches chacune étant dimensionnée par son temps de réponse pire cas. Le temps de réponse pire cas d'une tâche correspond, parmi tous les scénarios possibles, à la plus longue durée entre une activation de celle-ci et l'instant où l'exécution associée se termine. Dimensionner un système temps réel revient donc à déterminer le temps de réponse pire cas de chacune de ses tâches. Une fois le système dimensionné, nous vérifions si celui-ci répond aux contraintes temporelles qui lui sont imposées, c'est à dire si le temps de réponse pire cas de chaque tâche est inférieur ou égal à l'échéance fixée par le concepteur. Sachant que nous ne tolérons aucun dépassement, nous parlons de temps réel à contraintes strictes. Par opposition, le temps réel à contraintes relatives autorise le dépassement occasionnel des échéances. Ce chapitre s'intéresse aux systèmes temps réel composés exclusivement de tâches indépendantes, c'est à dire ne communiquant pas entre elles, et tels qu'aucune ressource, comme un périphérique de communication ou une mémoire, ne soit employée par plusieurs tâches. La section 2.a présente les différents modèles, les concepts ainsi que les notations usuelles. Les principaux algorithmes d'ordonnancement sont ensuite détaillés à la section 2.b. Afin de garantir un dimensionnement valable pour tout scénario possible, le système sera validé dans les pires conditions pour le respect des contraintes temporelles selon les scénarios décrits à la section 2.c. Les tests de validation à proprement parler, également appelés conditions de faisabilité, adaptés aux ordonnancements FP, FIFO et FP/FIFO, sont présentés à la section 2.d. Dans le cas d'un ordonnancement EDF, les conditions de faisabilité appropriées sont expliquées à la section 2.e. Enfin, la section 2.f récapitule les principales notions étudiées dans ce chapitre.

2.a Caractérisation d'une application temps réel

Dans de cette thèse, nous nous intéressons à des ensembles de tâches dont nous ignorons, a priori, les dates d'activation. Autrement dit, nous considérons toujours que tous les scénarios d'activations sont possibles. Dans ce contexte, la méthode la plus couramment utilisée pour mener à bien le dimensionnement d'un système temps réel respecte les étapes suivantes [31] :

- Identifier la classe du système à valider.
- Identifier les scénarios d'activations conduisant aux pires conditions pour le respect des contraintes temporelles également appelés scénarios pires cas.
- Etablir les conditions de faisabilité et les vérifier sur les scénarios pires cas précédents.

La classe d'un système temps réel est définie par le modèle de chacune de ses tâches, le modèle de contrainte temporelle considéré et le modèle de son ordonnancement. Les modèles précédents sont respectivement explicités aux sous sections 2.a.i 2.a.ii et 2.a.iii. Comme nous le constaterons à la section 2.c, la modélisation correcte du système est primordiale, tant au niveau de son ordonnancement pour déterminer les scénarios pires cas, qu'au niveau des tâches qui le constituent afin de mettre correctement en oeuvre les conditions de faisabilité. En effet, l'impact d'une tâche sur ses concurrentes varie selon ses caractéristiques. Cette thèse portant sur l'étude de nouvelles conditions de faisabilité tenant compte du système d'exploitation, cette étape de modélisation du système s'avère indispensable à leur application. Les concepts et notations utilisés en temps réel sont présentés à la sous section 2.a.iv. Puis, les trois techniques permettant l'obtention des conditions de faisabilité sont discutées à la sous section 2.a.v.

2.a.i Modèle temporel d'une tâche

Un système temps réel se modélise donc par un ensemble fini de tâches $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Cet ensemble de tâches découle directement de l'objectif de l'application, c'est à dire de son cahier des charges. Soit τ_i une tâche quelconque du système, τ_i peut être :

- **Périodique** : Lorsque ses activations sont périodiques et de période T_i.
- **Sporadique** : Lorsque deux activations successives de cette tâche sont espacées d'une durée supérieure ou égale à T_i.
- **Apériodique** : Lorsque ses activations surviennent à des moments aléatoires.

Chaque activation de la tâche τ_i étant suivie de son exécution, nous précisons que chaque nouvelle exécution définit une **instance**.

Une tâche est **non-concrète** lorsque nous n'avons aucune connaissance sur ses **instants d'activation**. Dans le cas contraire, la tâche sera dite **concrète**. Une tâche périodique est donc concrète si, et seulement si, nous connaissons la date de sa première activation.

Définissons, maintenant, la durée d'exécution et le temps de réponse pires cas d'une tâche :

- La **durée d'exécution pire cas**, ou WCET (Worst Case Execution Time), d'une tâche τ_i est notée C_i. Cette durée d'exécution pire cas correspond au pire temps d'exécution de la tâche τ_i seule, c'est à dire sans aucune préemption due à l'ordonnanceur ou à toute autre tâche. Plusieurs méthodes permettent de mesurer le WCET d'une tâche [50]. Soit par l'étude du code machine de la tâche et du processeur sur lequel celle-ci s'exécute. Soit par mesure directe sur une architecture réelle. Dans ce dernier cas, le résultat dépend de l'architecture : type de mémoire, périphériques, etc...
- Le **temps de réponse pire cas** r_i d'une tâche τ_i est la plus longue durée entre une activation de τ_i et la terminaison de l'exécution associée. La tâche τ_i pouvant être éventuellement interrompue pendant son exécution, au profit de tâches plus prioritaires, son temps de réponse pire cas dépend des caractéristiques des autres tâches du système ainsi que de la politique d'ordonnancement. Dans tous les cas, nous aurons obligatoirement $r_i \geq C_i$.

2.a.ii Modèle de contrainte temporelle

La contrainte temporelle, ou encore l'**échéance relative**, associée à une tâche τ_i, est notée D_i. L'objectif consistant à garantir le respect de la contrainte temporelle de chaque tâche présente dans le système, commençons par définir les différents modèles de contrainte temporelle :

- L'**échéance absolue de terminaison au plus tard** impose que toute tâche τ_i, activée à la date t, doit obligatoirement avoir terminé son exécution, au plus tard, à la date $t + D_i$.
- L'**échéance absolue de démarrage au plus tard** impose que toute tâche τ_i, activée à la date t, doit obligatoirement avoir débuté son exécution à la date $t + D_i$.
- Dans certains systèmes temps réel, l'échéance est modélisée par une **fonction de valeur du temps**, ou TVF (Time Value Function) [45] [1] [14] [56]. Lors de sa terminaison à la date t_t, la tâche τ_i, activée à la date t_a, est évaluée par la fonction de gain qui lui est associée $F_i(t_t - t_a)$. Cette fonction renvoie sa valeur maximale P si τ_i a terminé son exécution avant la date $t_a + D_i$, puis décroît vers le gain minimal N qui peut être négatif ou nul. Lorsque la fonction décroît brusquement, cela représente une échéance stricte à ne jamais dépasser. Une décroissance progressive de la fonction signifie, quant à elle, une échéance relative qui peut éventuellement être légèrement dépassée. L'ordonnancement des tâches devient ainsi un problème d'optimisation et revient à maximiser la somme des gains obtenus par l'ensemble des tâches. La figure 2.1 illustre, à l'aide d'un exemple, ce type de fonctions.

De plus, lorsque les caractéristiques T_i et D_i sont indépendantes, la tâche τ_i est dite à **échéance arbitraire**. Autrement, si ces deux caractéristiques sont égales, la tâche τ_i est dite à **échéance sur requête**.

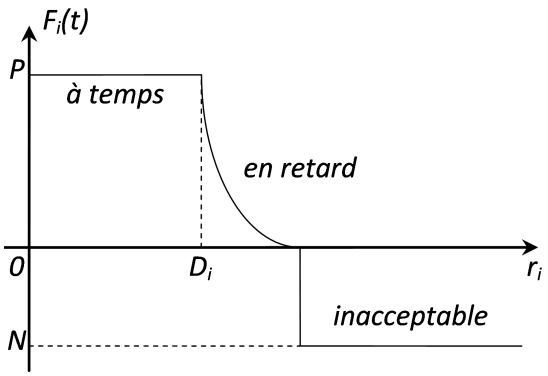

FIGURE 2.1 – Terminaison d'une tâche τ_i évaluée à l'aide d'une fonction TVF

2.a.iii Modèle d'ordonnancement

Le problème de l'**ordonnancement** consiste à choisir, si tant est qu'elle existe, une politique d'attribution du processeur aux tâches telle qu'aucune faute temporelle ne sera commise durant toute la vie du système. Deux approches peuvent être distinguées :

- L'**ordonnancement en ligne** : Un algorithme d'ordonnancement est implémenté. Les décisions d'ordonnancement sont alors prises pendant la vie du système.
- L'**ordonnancement hors ligne** : Une séquence valide, définissant l'ordre dans lequel les tâches doivent être exécutées, est déterminée avant le déploiement du système. Une fois chargée dans une table, cette séquence sera utilisée par un séquenceur tout au long de la vie du système.

Lors de l'étude d'un système temps réel, les temps de réponse pires cas obtenus dépendent fortement de la politique d'ordonnancement. Aussi, nous en présentons, dans cette sous section, les principales caractéristiques. Le concept de priorité, nécessaire à tout ordonnancement en ligne pour sélectionner la tâche qui a le plus besoin du processeur, est présenté au paragraphe 2.a.iii.1. Puis, le paragraphe 2.a.iii.2 introduit la notion de préemption. Enfin, les différents modèles d'invocation de l'ordonnanceur sont décrits au paragraphe 2.a.iii.3.

2.a.iii.1 Ordonnancements FP, DP et FP/DP

Afin de permettre à un ordonnanceur en ligne de déterminer l'ordre dans lequel les tâches doivent s'exécuter, une solution très classique consiste à attribuer une priorité à chacune d'entre elles. Lorsqu'il est invoqué, l'ordonnanceur sélectionne alors, selon l'algorithme HPF (Highest Priority First), la tâche la plus prioritaire. Nous distinguons deux types de priorités :

- **Priorité fixe** : La priorité de chaque tâche est fixée lors de la conception du système et demeure invariable tout au long de sa vie. Dorénavant, nous noterons **FP** (Fixed Priority) les algorithmes d'ordonnancement basés sur des priorités fixes avec l'algorithme en ligne HPF.

– **Priorité dynamique** : La priorité de chaque tâche est déterminée, en fonction de la politique d'ordonnancement, pendant l'exécution du système. Dorénavant, nous noterons **DP** (Dynamic Priority) les algorithmes d'ordonnancements basés sur des priorités dynamiques.
– **FP/DP** : Dans ce cas, l'ordonnancement est dit mixte, ou encore hybride, et utilise FP comme premier critère d'ordonnancement puis DP comme second critère lorsque plusieurs tâches possèdent la même priorité fixe.

2.a.iii.2 Ordonnancements préemptif, non préemptif et mixte

Dans un système temps réel, chaque tâche peut être **préemptive** ou **non prémptive**. L'ordonnanceur peut interrompre l'exécution d'une tâche préemptive au profit d'une autre tâche plus prioritaire. A l'inverse, l'ordonnanceur ne peut interrompre l'exécution d'une tâche non préemptive et doit attendre sa terminaison avant de débuter l'exécution de toute autre tâche. Ainsi, lorsqu'une tâche est activée, le début de son exécution peut être retardé par une tâche moins prioritaire à la condition que celle-ci soit non préemptive et en cours d'exécution. Nous parlons alors d'**effet non préemptif**. Dans ce cas, la tâche venant d'être activée doit attendre que la tâche moins prioritaire ait terminé son exécution avant de pouvoir débuter la sienne. Durant ce temps, la tâche moins prioritaire apparaît plus prioritaire. En conséquence, nous pouvons dire que l'effet non préemptif engendre une **inversion de priorité**.

Lorsque toutes les tâches de l'application sont préemptives, l'ordonnancement est alors dit **préemptif**. Dans ce cas, dès qu'une tâche plus prioritaire que la tâche en cours d'exécution est activée, celle-ci se voit attribuer le processeur. La figure 2.2 illustre l'ordonnancement FP préemptif de trois tâches τ_1, τ_2 et τ_3. Nous précisons que la tâche τ_1 est plus prioritaire que la tâche τ_2 qui est elle-même plus prioritaire que la tâche τ_3. Nous observons alors que la tâche τ_2 interrompt l'exécution de la tâche τ_3. De même, une fois activée, la tâche τ_1 récupère immédiatement le processeur au détriment de la tâche τ_2.

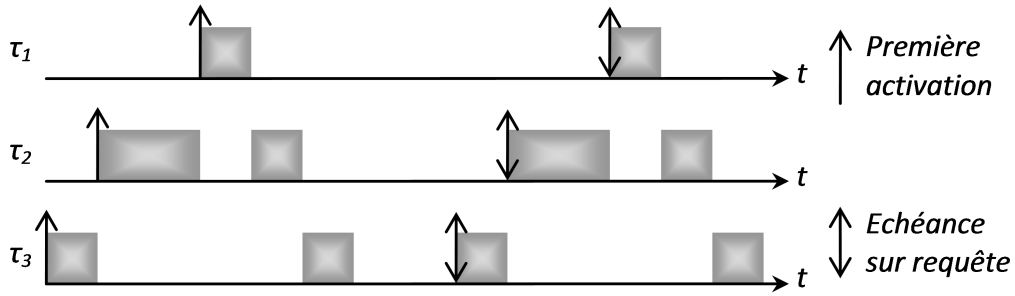

FIGURE 2.2 – Illustration d'un ordonnancement FP préemptif

Si toutes les tâches de l'application sont non préemptives, l'ordonnancement est alors dit **non préemptif**. Dans ce cas, même si une tâche plus prioritaire était activée, la tâche en cours d'exécution ne perdrait pas l'utilisation du processeur à son profit. La figure 2.3 illustre l'ordonnancement FP non préemptif de trois tâches τ_1, τ_2 et τ_3 dont les priorités demeurent inchangées par rapport au cas précédent. Nous constatons que la tâche τ_2,

étant activée après la tâche τ_3, bien que plus prioritaire que celle-ci, doit attendre la fin de son exécution avant d'entamer la sienne. De la même façon, la tâche τ_1 est retardée par la tâche τ_2. Nous observons donc un premier effet non préemptif de τ_3 sur τ_2, puis un second de τ_2 sur τ_1.

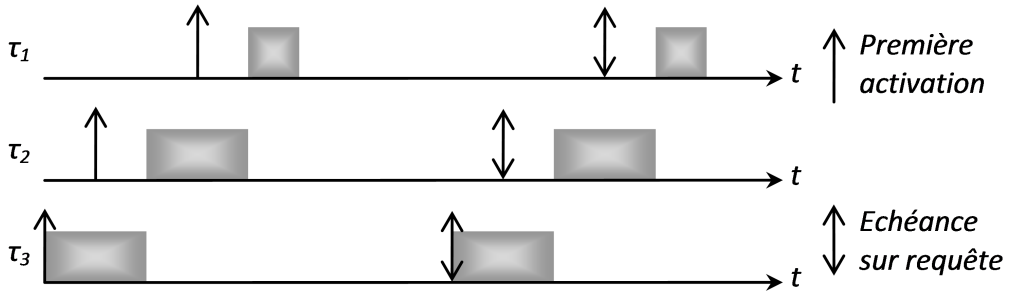

FIGURE 2.3 – Illustration d'un ordonnancement non préemptif

Enfin, dans une même application, certaines tâches peuvent être préemptives et d'autres non. Dans ce cas, l'ordonnancement est dit **mixte**. Lorsqu'une tâche, plus prioritaire que la tâche en cours d'exécution τ_c, est activée, celle-ci récupère ou non le processeur selon que la tâche τ_c soit préemptive ou non. La figure 2.4 illustre l'ordonnancement FP mixte de trois tâches τ_1, τ_2 et τ_3 ayant, là encore, des priorités identiques par rapport aux cas précédents. De la même façon qu'avec l'ordonnancement FP non préemptif, la tâche non préemptive τ_3 retarde la tâche τ_2 pourtant plus prioritaire. Par contre, l'exécution de la tâche préemptive τ_2 est interrompue par la tâche τ_1. Ainsi, nous n'observons, cette fois, qu'un seul effet non préemptif de la tâche τ_3 sur la tâche τ_2.

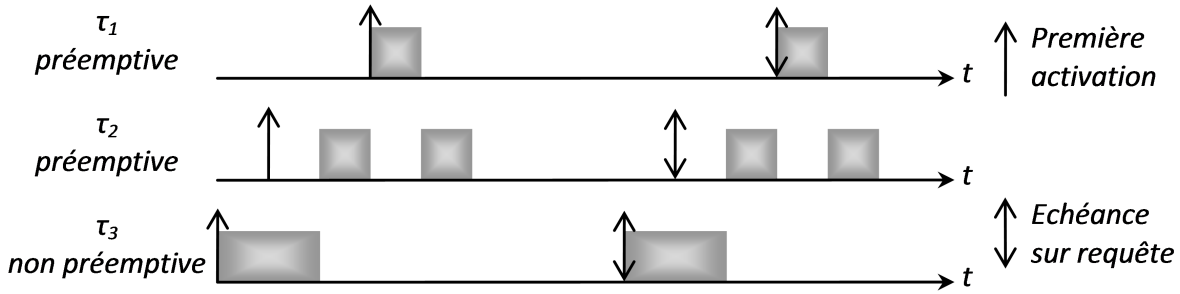

FIGURE 2.4 – Illustration d'un ordonnancement mixte

2.a.iii.3 Modèles d'invocation d'un ordonnanceur

Dans tous les cas, une fois activée, l'exécution d'une tâche ne débutera qu'à partir du moment où l'ordonnanceur l'y aura autorisée. Ainsi, les dates auxquelles l'ordonnanceur est invoqué impactent considérablement le comportement du système. Nous exposons donc maintenant les deux modèles d'invocation de l'ordonnanceur :

- **Guidé par les événements** : L'ordonnanceur est invoqué sur réception d'un événement : demande d'activation d'une tâche, libération d'une ressource, réception d'un message, interruption, etc...

– **Guidé par le temps** : Les instants d'invocation de l'ordonnanceur sont indépendants des événements qui se produisent dans le système et gérés par le système lui-même.

Un ordonnancement guidé par le temps s'appuie, le plus souvent, sur une invocation périodique ce qui implique des retards sur la prise en compte des événements. La valeur de cette période, fixée à $1ms$ dans la plupart des sytèmes temps réel, est naturellement limitée par la durée d'exécution pire cas de l'ordonnanceur [16].

2.a.iv Concepts et notations

Dans la suite de cette analyse temps réel, le temps sera, le plus souvent, considéré comme **discret** : toutes les dates ainsi que les paramètres des tâches seront entiers et exprimés dans la même unité. L'article [7] montre l'intérêt de ce choix selon les faits suivants :

– La plupart des ordonnanceurs utilisent une horloge comme référence temporelle.
– Si tous les paramètres temporels sont entiers et exprimés dans la même unité, alors il existe une solution à base d'ordonnancement continu si, et seulement si, il en existe une en temps discret.

Ainsi, lorsque les paramètres des tâches sont tous entiers et exprimés dans la même unité, nous considérons sans perte de généralité que les dates d'invocation de l'ordonnanceur, exprimées dans cette même unité, sont également entières.

Soit $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ un ensemble constitué de n tâches, chacune d'entre elles étant périodique ou sporadique :

– $\forall x \in \Re$, $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à x. Soit une tâche périodique τ_i, $\lceil \frac{\omega}{T_i} \rceil$ correspond au nombre d'activations de la tâche τ_i sur l'intervalle $[0, \omega[$, la première ayant lieu à la date 0.
– $\forall x \in \Re$, $\lfloor x \rfloor$ désigne le plus grand entier inférieur ou égal à x. Soit une tâche périodique τ_i, $1 + \lfloor \frac{\omega}{T_i} \rfloor$ correspond au nombre d'activations de la tâche τ_i sur l'intervalle $[0, \omega]$, la première ayant lieu à la date 0.
– Un jeu de tâches périodiques est dit **synchrone** si il existe un instant où toutes ses tâches sont activées simultanément, nous parlons alors d'**instant critique**. Autrement, il sera dit **asynchrone**.
– Le **scénario d'activation synchrone** correspond au cas où la première activation de chaque tâche est synchrone à la date 0.
– Des tâches sont **indépendantes** lorsqu'elles ne sont définies que par leurs paramètres temporels. Au contraire, lorsque des tâches communiquent entre elles, des **contraintes de précédence** sont introduites par ces intéractions.
– La **charge de travail** requise par une tâche τ_i dans l'intervalle $[0, \omega[$, si sa première activation arrive à la date 0, vaut : $\lceil \frac{\omega}{T_i} \rceil C_i$.
– De même, la charge de travail requise par une tâche τ_i dans l'intervalle $[0, \omega]$, si sa première activation arrive à la date 0, vaut : $\left(1 + \lfloor \frac{\omega}{T_i} \rfloor\right) C_i$.

- Par convention, $\max^*_\tau(x)$ désigne la valeur maximale du paramètre x dans τ si $\tau \neq \phi$ et 0 sinon.
- Dans le cas d'un ordonnancement FP, la priorité fixe d'une tâche τ_i est symbolisée par un paramètre, noté P_i, qui lui est inversement proportionnel.
- Dans le cas d'un ordonnancement DP, la priorité dynamique d'une tâche τ_i, activée à la date a et évaluée par l'ordonnanceur à la date t, est symbolisée par un paramètre, noté $P_i(t, a)$, qui lui est inversement proportionnel.

2.a.v Les techniques d'analyse

La validation d'un système temps réel revient à montrer que chacune de ses tâches respectera toutes ses échéances temporelles tout au long de la vie dudit système. Pour ce faire, trois techniques existent :

- La première étudie le **facteur d'utilisation du processeur** [38] [25]. Le facteur d'utilisation est le pourcentage de temps que le processeur passe à exécuter l'ensemble des n tâches du système, et vaut $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Comme nous le verrons à la section 2.d, dans certains cas, ce facteur d'utilisation suffit à valider l'ordonnançabilité d'un ensemble de tâches. Dans tous les cas, un jeu de tâches ne peut être ordonnançable que si $U \leq 1$.
- La seconde se base sur la **demande processeur** (Processor Demand Analysis) [8] [52]. Notée $h(t)$, la demande processeur est le temps d'exécution requis pour exécuter, dans le scénario d'activation synchrone, toutes les tâches censées être activées et terminées dans l'intervalle $[0, t]$: $h(t) = \sum_{\tau_i \in \tau / D_i \leq t} \left(1 + \lfloor \frac{t-D_i}{T_i} \rfloor\right) C_i$. Le système n'est alors pas ordonnançable s'il existe une date t telle que $h(t) > t$. Cette méthode aboutit à des tests d'ordonnançabilité [33] [28] [5] et limite l'étude à quelques instants d'une **période d'activité** du processeur. Une période d'activité du processeur définissant un intervalle de temps durant lequel le processeur est utilisé en permanence.
- La dernière vient de l'**analyse des temps de réponse** (Response Time Analysis) [29] [32]. Cette méthode nécessite le calcul du temps de réponse pire cas de chaque tâche du système temps réel étudié. Ce système est ordonnançable si, et seulement si, chaque tâche τ_i de celui-ci vérifie $r_i \leq D_i$.

Les deux dernières techniques s'appuient sur l'analyse des périodes d'activité du processeur. Ainsi, ces deux techniques sont parfois confondues sous le terme "analyse de la demande de temps" (Time Demand Analysis) [37]. Cependant, l'analyse des temps de réponse apporte un diagnostic plus détaillé du système que l'analyse de la demande processeur. Dans le cas où le système ne serait pas ordonnançable, ce diagnostic permet d'en cerner la raison et éventuellement de la corriger.

2.b Algorithmes d'ordonnancement

Comme nous l'avons dit à la sous section 2.a.iii, dans un système temps réel, chaque tâche se voit attribuer une priorité afin qu'à chacune de ses invocations l'ordonnanceur soit en mesure d'élire la tâche actuellement la plus prioritaire parmi celles activées et de lui accorder l'utilisation du processeur.

Un algorithme d'ordonnancement est dit **optimal** par rapport à une classe de problème d'ordonnancement si il trouve toujours une solution à un problème de cette classe lorsqu'il en existe une. Ainsi, si un algorithme d'ordonnancement optimal pour une classe donnée ne trouve pas de solution à un problème de cette classe, alors celui-ci ne peut être résolu.

Ce paragraphe présente les algorithmes d'ordonnancement usuels. Les algorithmes à priorités fixes sont explicités en premiers à la sous section 2.b.i, viennent ensuite les algorithmes à priorités dynamiques à la sous section 2.b.ii, et enfin les algorithmes hybrides à la sous section 2.b.iii.

2.b.i Algorithmes à priorités fixes

Cette sous section présente les trois méthodes courantes d'attribution hors ligne de priorités fixes à un ensemble de n tâches noté τ. Conformément à la notation définie à la sous section 2.a.iv, la priorité de la tâche τ_i est inversement proportionnelle au paramètre P_i qui lui est associé.

2.b.i.1 Rate Monotonic

L'algorithme RM (Rate Monotonic) a été présenté par Liu et Layland en 1973 [38]. A travers cet algorithme, chaque tâche τ_i se verra attribuer une priorité inversement proportionnelle à sa période, c'est à dire un paramètre P_i proportionnel à T_i. Ainsi, la priorité maximale correspondra à la période la plus courte. Dans le cas où plusieurs tâches possèdent la même période, leur ordonnancement sera alors arbitraire (FIFO pourra être utilisé par exemple, voir le paragraphe 2.b.ii.3). Si le système ne comporte que des tâches indépendantes à échéances sur requête alors l'algorithme RM est optimal pour obtenir une séquence valide.

Optimalité *L'algorithme RM est optimal parmi les algorithmes à priorités fixes pour l'ordonnancement de tâches à échéance sur requête, périodiques ou sporadiques, préemptives et indépendantes. Comme le montre L.George dans sa thèse [23], l'algorithme RM n'est pas optimal en contexte non préemptif. De plus, son optimalité n'est pas générale ; autrement dit, un système non ordonnançable avec l'algorithme RM peut être ordonnancé à l'aide d'un algorithme à priorités dynamiques.*

De plus, si l'ensemble τ comporte exclusivement des tâches à échéances sur requête, l'algorithme RM vérifie le **théorème de l'instant critique** : n'importe quelle tâche τ_i de cet ensemble obtiendra son temps de réponse pire cas si elle est activée en même temps que

toutes les tâches plus prioritaires qu'elle. Ainsi, lorsque toutes les tâches sont activées simultanément à la date 0, si aucune faute temporelle ne se produit sur l'intervalle de temps $[0, \max_{\tau_i \in \tau}(T_i)]$ alors aucune faute ne se produira durant toute la vie du système quelles que soient les dates d'activation des tâches. A partir de ce résultat, L.C.Liu et W.Layland ont établi une condition suffisante à la validation d'un système basé sur l'algorithme RM :

Proposition 2.1 *Un système de n tâches périodiques ou sporadiques préemptives indépendantes à échéances sur requête est ordonnançable par RM si* $U \leq n(2^{\frac{1}{n}} - 1)$.

L'avantage de cette condition suffisante réside en sa simplicité. Cependant, une condition nécessaire et suffisante, plus complexe, est étudiée dans les papiers [29] [33]. Considérons, à titre d'exemple, le système constitué de trois tâches, préemptives et indépendantes, dont les caractéristiques temporelles sont décrites dans le tableau 2.1 :

Tâche	*C*	*D=T*
τ_1	3	11
τ_2	4	15
τ_3	1	5

TABLE 2.1 – Algorithme RM : caractéristiques des tâches de l'exemple

La charge de travail totale pour ce système vaut $3/11 + 4/15 + 1/5 = 0,74$. Or, selon la proposition 2.1, ce système étant composé de trois tâches est ordonnançable avec l'algorithme RM si sa charge de travail ne dépasse pas $3(2^{1/3} - 1) = 0,78$. Ainsi, ce système est bien ordonnançable par l'algorithme RM. La figure 2.5 illustre ce résultat lorsque toutes les tâches sont activées simultanément à la date 0 :

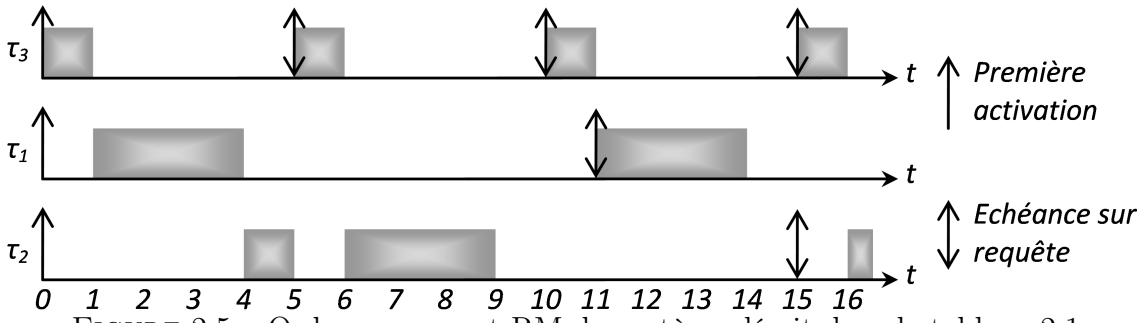

FIGURE 2.5 – Ordonnancement RM du système décrit dans le tableau 2.1

Nous n'observons aucune faute temporelle sur l'intervalle $[0, 15]$ ce qui confirme bien la conclusion précédente. Remarquons au passage que si nous avions $C_2 = 5$, la charge de travail totale du système serait égale à $0,81$. La condition suffisante ne serait alors plus validée mais le système resterait ordonnançable avec l'algorithme RM.

2.b.i.2 Deadline Monotonic

L'algorithme RM présenté précédemment n'est optimal qu'avec des tâches à échéances sur requête. Aussi, J.Y.T.Leung et J.Whitehead ont amélioré ce dernier afin d'accepter

des échéances plus courtes [36] telles que, $\forall \tau_i \in \tau, D_i \leq T_i$. Ce nouvel algorithme attribue à chaque tâche τ_i une priorité inversement proportionnelle à son échéance, c'est à dire un paramètre P_i propotionnel à D_i. Ainsi cette méthode d'attribution des priorités s'appelle l'algorithme DM (Deadline Monotonic). Comme pour l'algorithme RM, dans le cas où plusieurs tâches possédent la même échéance, leur ordonnancement sera alors arbitraire (FIFO pourra être utilisé par exemple, voir le paragraphe 2.b.ii.3). En présence de tâches à échéances sur requête, les algorithmes RM et DM sont équivalents. Dans le cas contraire, l'algorithme DM présente une attribution optimale :

Optimalité *L'algorithme DM est optimal parmi les algorithmes à priorités fixes pour l'ordonnancement d'un ensemble de tâches périodiques ou sporadiques préemptives indépendantes et telles que,* $\forall \tau_i \in \tau, D_i \leq T_i$. *Comme le montre L.George dans sa thèse [23] ainsi que dans le papier [24], l'algorithme DM n'est optimal en contexte non préemptif que si* $\forall \tau_i \in \tau, D_i \leq T_i$ *et* $\forall \tau_i, \tau_j \in \tau, C_i \leq C_j \Rightarrow D_i \leq D_j$. *De plus, comme pour l'algorithme RM, l'optimalité de l'algorithme DM n'est pas générale.*

Comme l'algorithme RM, si, $\forall \tau_i \in \tau, D_i \leq T_i$, l'ordonnancement DM vérifie le théorème de l'instant critique. Ainsi, lorsque toutes les tâches sont activées simultanément à la date 0, si aucune faute temporelle ne se produit sur l'intervalle de temps $[0, \max_{\tau_i \in \tau}(D_i)]$ alors aucune faute ne se produira durant toute la vie du système quelles que soient les dates d'activation des tâches. Ceci permet d'écrire une condition suffisante d'ordonnançabilité. Une condition nécessaire et suffisante est présentée dans [4] [17] mais, comme dans le cas de l'algorithme RM, celle-ci s'avère plus complexe que la condition suffisante.

Proposition 2.2 *Un système de n tâches périodiques ou sporadiques préemptives indépendantes et telles que,* $\forall \tau_i \in \tau$, $D_i \leq T_i$ *est ordonnançable par DM si* $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1)$.

Nous illustrons maintenant l'algorithme DM à l'aide d'un système à trois tâches, préemptives et indépendantes, dont les caractéristiques temporelles figurent au tableau 2.2 :

Tâche	*C*	*D*	*T*
τ_1	3	10	12
τ_2	2	8	15
τ_3	1	5	5

TABLE 2.2 – Algorithme DM : caractéristiques des tâches de l'exemple

Ce système vérifiant la proposition 2.2 ($3/10 + 2/8 + 1/5 = 0,75 \leq 0,78$) est bien ordonnançable par l'algorithme DM. La figure 2.6 illustre l'exécution de ce système dans le cas où toutes les tâches sont activées simultanément à la date 0. Nous n'observons aucune faute temporelle sur l'intervalle $[0, 10]$ ce qui confirme bien la conclusion précédente. Remarquons au passage que si nous avions $C_1 = 4$, la condition suffisante ne serait plus vérifiée mais le système resterait ordonnançable avec l'algorithme DM.

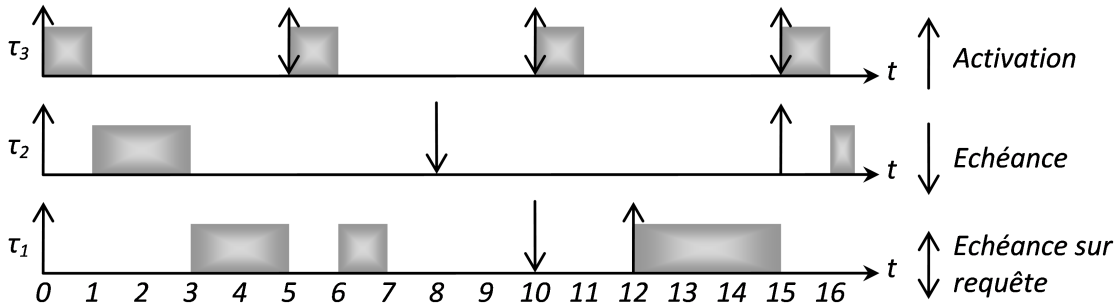

FIGURE 2.6 – Ordonnancement DM du système décrit dans le tableau 2.2

2.b.i.3 Algorithme d'Audsley

Aucun des deux algorithmes présentés précédemment n'est optimal en général parmi les algorithmes à priorités fixes. En effet, en contexte préemptif, l'algorithme RM n'est optimal que pour les ensembles de tâches à échéances sur requête. De même, la condition $\forall \tau_i \in \tau, D_i \leq T_i$ doit être vérifiée pour que l'algorithme DM soit optimal sur l'ensemble τ. Nous étudions maintenant l'algorithme proposé par N.C.Audsley [3] qui comble cette lacune et apporte toujours une solution valide, si tant est qu'il en existe une, à base de priorités fixes sans aucune contrainte sur les caractéristiques des tâches.

Optimalité *La méthode d'attribution des priorités fixes proposée par N.C.Audsley [3] est optimale pour des ensembles de tâches généraux, c'est à dire tels que, $\forall \tau_i \in \tau$, T_i et D_i sont parfaitement indépendants. De plus, cette méthode est optimale pour l'ordonnancement préemptif de tâches périodiques ou sporadiques. Comme le montre L.George dans sa thèse [23], cette méthode demeure valable en contexte non préemptif. Finalement, N.C.Audsley [3] propose une méthode d'attribution des priorités fixes optimale.*

L'algorithme itératif d'Audsley utilise une fonction nommée *cherche-si-faisable(τ, P)* qui cherche dans l'ensemble τ une tâche τ_i qui respecterait son échéance temporelle avec son paramètre P_i égal à P alors que toutes les autres tâches de τ seraient plus prioritaires qu'elle. Cette fonction s'appuie sur le calcul du temps de réponse pire cas d'une tâche que nous détaillons à la section 2.d. Si aucune tâche n'est trouvée, la fonction retourne 0 ce qui signifie que le système n'est pas ordonnançable avec des priorités fixes. Au contraire, si plusieurs tâches respectent leur échéance avec ce paramètre P, la fonction retourne l'une d'entre elles arbitrairement sans remettre en cause l'optimalité de l'attribution des priorités. Dans la logique de la fonction *cherche-si-faisable(τ, P)*, cet algorithme itératif cherche à attribuer, au fur et à mesure des itérations, une priorité de plus en plus forte, c'est à dire un paramètre P de plus en plus faible. Nous décrivons maintenant l'algorithme en question :

```
τ = {τ1, ..., τn} : ensemble de tâches ;
P ← n : entier ;  i : entier ;
fin← faux : booléen ;
Tant que (τ ≠ ∅ et fin=faux) faire
    i=cherche-si-faisable(τ,P) ;
    Si (i ≠ 0) Alors
        attribue-parametre(i,P) ; [on fixe le paramètre Pi de la tâche τi à P]
        τ = τ−{τi} ; [on retire τi de l'ensemble τ]
        P ← P-1 ; [on passe à la priorité supérieure]
    Sinon
        fin=vrai ;
    Fin Si
Fait
```

Algorithme 2.1: Attribution optimale des priorités - Méthode d'Audsley

Soulignons que la non faisabilité d'un ordonnancement en contexte préemptif n'implique pas que le problème ne soit pas faisable en contexte non préemptif et inversement [24]. Nous illustrons maintenant cette méthode d'attribution des priorités fixes à l'aide d'un système à trois tâches, préemptives et indépendantes, dont les caractéristiques temporelles sont spécifiées au tableau 2.3 :

Tâche	C	D	T
τ_1	1	9	4
τ_2	3	8	8
τ_3	3	7	10

TABLE 2.3 – Algorithme d'Audsley : caractéristiques des tâches de l'exemple

Lors de la première itération, la fonction *cherche-si-faisable* recherche, parmi l'une des trois tâches précédentes, une tâche qui respecterait toujours sa contrainte temporelle alors que les deux autres seraient plus prioritaires qu'elle. La figure 2.7 montre que la tâche τ_3 ne respecte pas son échéance lorsque les tâches τ_1 et τ_2 sont plus prioritaires qu'elle. Cette même figure montre que la tâche τ_1 ne commet, quant à elle, aucune faute lorsqu'elle est moins prioritaire que les tâches τ_2 et τ_3. De même, la tâche τ_2 respecte toujours son échéance même en étant la moins prioritaire. A la fin de cette première itération, la plus faible priorité est arbitrairement attribuée à la tâche τ_1 dont le paramètre P_1 est fixé à 3.

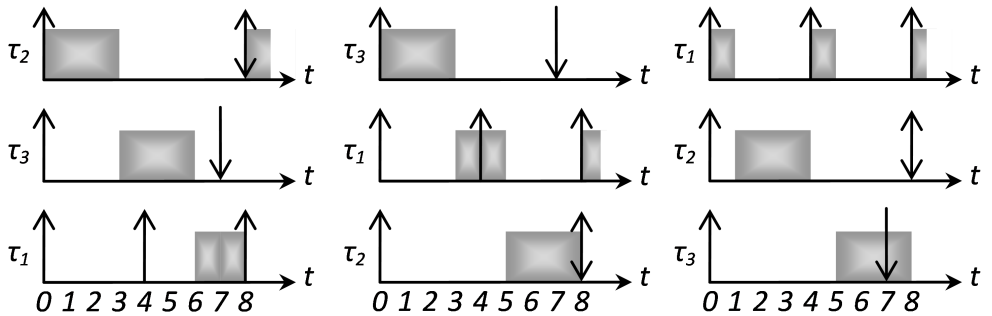

FIGURE 2.7 – Algorithme d'Audsley : illustration de la première itération

Lors de la seconde itération, la fonction *cherche-si-faisable* recherche, parmi les tâches τ_2 et τ_3, une tâche respectant toujours sa contrainte temporelle tout en étant moins prioritaire que l'autre. La figure 2.8 montre que, quelle que soit la moins prioritaire, les tâches τ_2 et τ_3 ne commettent aucune faute. Ainsi, la tâche τ_2 se verra arbitrairement attribuer une priorité plus faible que celle de la tâche τ_3. Pour ces deux tâches, nous obtenons $P_2 = 2$ et $P_3 = 1$.

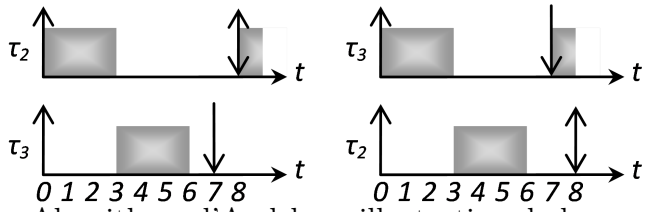

FIGURE 2.8 – Algorithme d'Audsley : illustration de la seconde itération

Finalement, d'après la méthode d'Audsley, l'ensemble de tâches décrit au tableau 2.3 est ordonnançable avec les paramètres $P_1 = 3$, $P_2 = 2$ et $P_3 = 1$. La figure 2.9 illustre l'exécution de ce système dans le cas où toutes les tâches sont activées simultanément à la date 0 ·

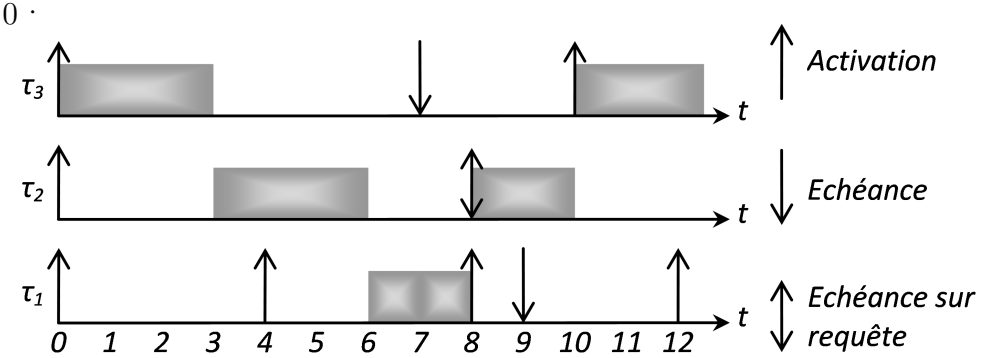

FIGURE 2.9 – Ordonnancement FP du système décrit dans le tableau 2.3

2.b.ii Algorithmes à priorités dynamiques

Cette sous section présente les principales méthodes d'attribution en ligne de priorités dynamiques à un ensemble de n tâches noté τ. Bien que leur mise en oeuvre soit plus complexe, les algorithmes à priorités dynamiques présentent l'intérêt de pouvoir ordonnancer davantage de systèmes que ceux à priorités fixes. Conformément à la notation définie à la sous section 2.a.iv, la priorité de la tâche τ_i, activée à la date a, évaluée par l'ordonnanceur à la date t est inversement proportionnelle au paramètre $P_i(t, a)$ qui lui est associé.

2.b.ii.1 Earliest Deadline First

Parmi les algorithmes à priorités dynamiques, EDF (Earliest Deadline First) est vraisemblablement le plus populaire. Comme présenté dans [38], cet algorithme repose sur le principe qu'à chacune de ses invocations l'ordonnanceur élit la tâche dont l'échéance est la plus proche. Le livre [53] fournit une présentation détaillée de cet algorithme.

Optimalité *L'algorithme EDF est optimal pour l'ordonnancement des tâches périodiques ou sporadiques en contexte préemptif [20] ainsi qu'en contexte non préemptif non concret [27]. Cette optimalité n'est plus vraie en contexte non préemptif concret [27].*

Pour des tâches à échéances sur requête, l'algorithme EDF présente l'avantage de posséder une condition nécessaire et suffisante d'ordonnançabilité simple à mettre en oeuvre. Cette condition a été établie par L.C.Liu et W.Layland [38] :

Proposition 2.3 *Un ensemble de tâches à échéances sur requête est ordonnançable par EDF si, et seulement si, sa charge de travail totale U est inférieure ou égale à* 1.

Pour des tâches quelconques, l'article [35] montre une condition suffisante ainsi qu'une condition nécessaire :

Proposition 2.4 *Un ensemble de tâches τ, tel que, $\forall \tau_i \in \tau$, T_i et D_i sont parfaitement indépendants, est ordonnançable par EDF si $\sum_{\tau_i \in \tau} \frac{C_i}{D_i} \leq 1$, et seulement si sa charge totale de travail est inférieure ou égale à* 1.

Illustrons cet algorithme à l'aide du système constitué des deux tâches, préemptives et indépendantes, à échéances sur requête décrites au tableau 2.4.

Tâche	C	D=T
τ_1	4	8
τ_2	2	6

TABLE 2.4 – Algorithme EDF : caractéristiques des tâches de l'exemple

La charge de travail totale de ce système valant $4/8 + 2/6 = 0,83$, conformément à la proposition 2.3, celui-ci devrait être ordonnançable par EDF.

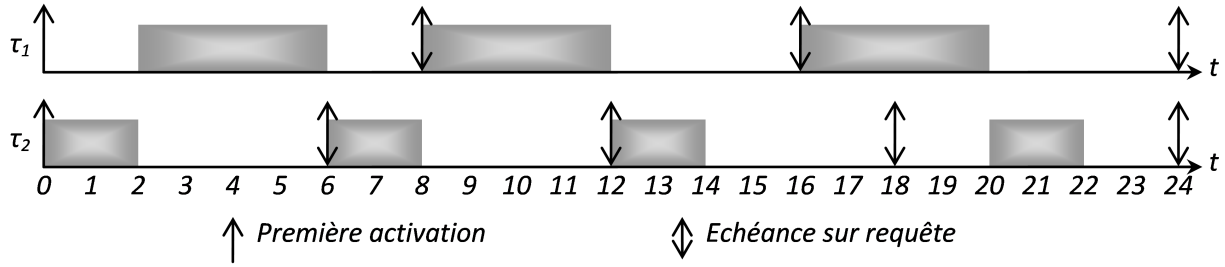

FIGURE 2.10 – Ordonnancement EDF du système décrit dans le tableau 2.4

Comme nous l'observons sur la figure 2.10, l'algorithme EDF ordonnance correctement ce système ce qui confirme la conclusion précédente. De plus, nous observons que les deux tâches sont réactivées simultanément à la date 24, le système se retrouve alors dans son état de départ. Ainsi, nous pouvons conclure que ce système ne rencontrera jamais aucune faute temporelle avec l'ordonnancement EDF.

2.b.ii.2 Least Laxity First

Avec l'algorithme LLF (Least Laxity First), introduit par A.K.Mok et M.L.Dertouzos [44] [43], à chacune de ses invocations l'ordonnanceur élit la tâche dont la laxité est la plus faible. Comme illustrée sur la figure 2.11, pour une tâche τ_i activée à la date a, la laxité est définie à la date t par : $l_i(t) = a + D_i - t - C_i(t)$ où $C_i(t)$ est la durée d'exécution restante pour la tâche τ_i à la date t. Autrement dit, pour une tâche τ_i, à la date t, sa laxité $l_i(t)$ représente le temps libre qui lui reste avant sa prochaine échéance.

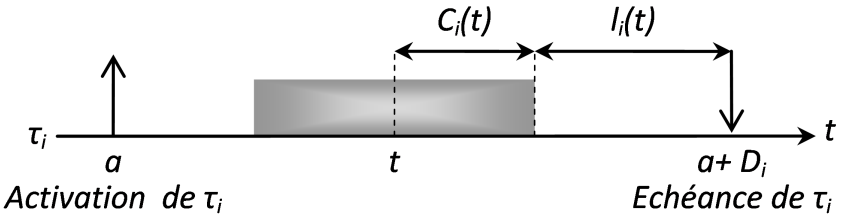

FIGURE 2.11 – Illustration de la laxité d'une tâche τ_i, activée à la date a, à la date t

Optimalité *Comme l'algorithme EDF, LLF est optimal pour l'ordonnancement de tâches périodiques ou sporadiques en contexte préemptif [43]. Par contre, cet algorithme n'est pas optimal en contexte non préemptif [23].*

L'ouvrage [19] montre que les conditions d'ordonnançabilité pour l'algorithme LLF sont les mêmes que pour EDF. Dans le cas de tâches à échéances sur requête, nous aurons donc une condition nécessaire et suffisante :

Proposition 2.5 *Un ensemble de tâches à échéances sur requête est ordonnançable par LLF si, et seulement si, sa charge de travail totale U est inférieure ou égale à* 1.

Pour des tâches quelconques nous aurons à nouveau une condition suffisante ainsi qu'une condition nécessaire :

Proposition 2.6 *Un ensemble de tâches τ, tel que, $\forall \tau_i \in \tau$, T_i et D_i sont parfaitement indépendants, est ordonnançable par LLF si $\sum_{\tau_i \in \tau} \frac{C_i}{D_i} \leq 1$, et seulement si sa charge totale de travail est inférieure ou égale à* 1.

L'algorithme LLF présente l'inconvénient, lorsque plusieurs tâches possèdent la même laxité, d'engendrer un grand nombre de changements de contexte ce qui explique qu'il soit aussi peu utilisé dans le cas monoprocesseur. Pour illustrer ceci, la figure 2.12 montre l'ordonnancement avec l'algorithme LLF du système présenté au tableau 2.4. L'ordonnançabilité de ce système ayant déjà été montrée avec l'algorithme EDF, ce résultat vaut également pour l'algorithme LLF. Sans surprise, l'ordonnancement de ce système avec l'algorithme LLF n'engendre aucune faute temporelle. Cependant, par comparaison avec la figure 2.10, nous observons que le nombre de changements de contexte requis par LLF sur l'intervalle $[0, 24[$ a quasiment doublé par rapport à EDF.

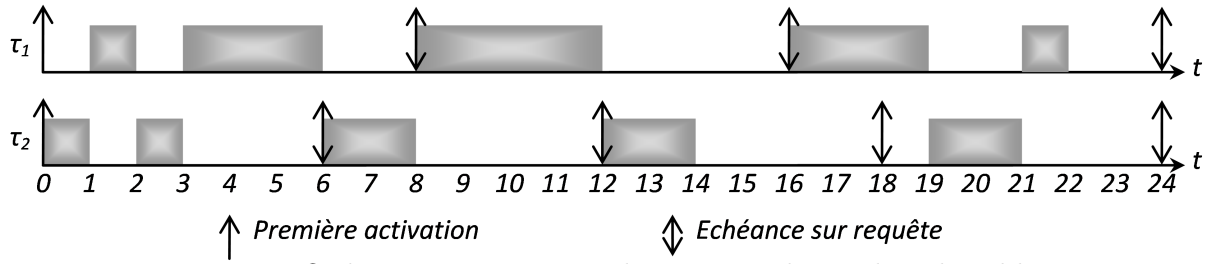

FIGURE 2.12 – Ordonnancement LLF du système décrit dans le tableau 2.4

2.b.ii.3 First In First Out

L'ordonnancement FIFO (First In First Out) revient à exécuter les tâches chacune leur tour dans l'ordre chronologique de leurs activations. Ainsi l'ordonnancement FIFO peut être considéré comme un algorithme à priorité dynamique dans lequel la priorité d'une tâche découle directement de sa date d'activation.

Optimalité *Dans l'article [26], J.Jackson montre que l'ordonnancement FIFO est optimal pour minimiser le temps de réponse pire cas d'un ensemble de tâches de même importance. FIFO est également optimal pour l'ordonnancement de tâches non concrètes lorsque celles-ci possèdent les mêmes contraintes temporelles, FIFO se comporte alors exactement comme l'algorithme EDF. Par contre, FIFO n'est pas optimal pour l'ordonnancement de tâches périodiques ou sporadiques.*

Illustrons cet ordonnancement à l'aide du système constitué des trois tâches indépendantes dont les caractéristiques temporelles sont décrites au tableau 2.5. Chaque tâche τ_i de cet ensemble possède un paramètre a_i correspondant à la date de sa première activation.

Tâche	a	C	$D{=}T$
τ_1	0	3	8
τ_2	1	2	8
τ_3	2	1	8

TABLE 2.5 – Algorithme FIFO : caractéristiques des tâches de l'exemple

La figure 2.13 montre l'exécution du système précédent avec l'ordonnancement FIFO. Nous observons qu'aucune tâche ne commet de faute temporelle. Les trois périodes étant égales le motif représenté se répète indéfiniment, ainsi le système respectera toujours ses échéances. De plus, les trois tâches ayant les mêmes contraintes temporelles, nous remarquons que FIFO se comporte bien comme EDF dans ce cas.

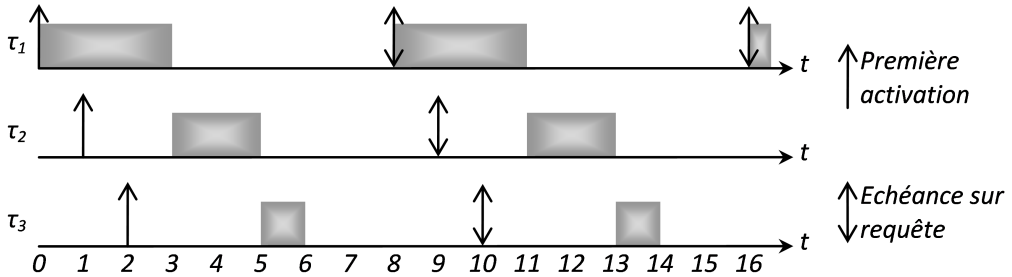

FIGURE 2.13 – Ordonnancement FIFO du système décrit dans le tableau 2.5

2.b.ii.4 Round Robin

L'ordonnancement RR (Round Robin) est défini dans la norme Posix 1003.1.b. Cette méthode consiste à placer toutes les tâches activées dans une file d'attente, le temps étant divisé en créneaux égaux, à la fin de chacun d'eux l'ordonnanceur se contente d'attribuer le processeur à la tâche située au sommet de la file d'attente puis de la replacer à la fin de celle-ci dès que le créneau se termine et ainsi de suite. Les tâches s'exécutent donc à tour de rôle sur les créneaux successifs.

Optimalité *Dans sa thèse [46], N.Navet montre qu'un système non ordonnançable avec des priorités fixes peut être ordonnancé à l'aide de l'algorithme RR. Cependant, l'algorithme RR n'est pas optimal pour l'ordonnancement de tâches périodiques ou sporadiques.*

Illustrons cet ordonnancement à l'aide du système constitué des trois tâches dont les caractéristiques temporelles sont données au tableau 2.6. Les trois tâches τ_1, τ_2 et τ_3 étant activées simultanément à la date 0, nous supposons qu'à cette date celles-ci sont respectivement placées en première, seconde et dernière position dans la file d'attente.

Tâche	*C*	*D=T*
τ_1	6	14
τ_2	3	14
τ_3	4	14

TABLE 2.6 – Algorithme RR : caractéristiques des tâches de l'exemple

La figure 2.14 montre l'exécution du système précédent avec l'ordonnancement RR. Nous observons qu'aucune tâche ne commet de faute temporelle, les trois périodes étant égales, le système respectera toujours ses échéances. De plus, nous remarquons que le temps écoulé entre l'activation d'une tâche et sa première allocation dépend du nombre de tâches déjà présentes dans la file d'attente.

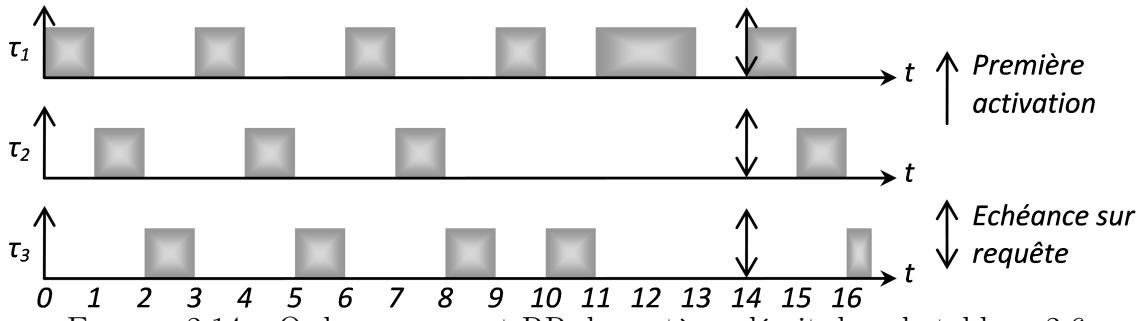

FIGURE 2.14 – Ordonnancement RR du système décrit dans le tableau 2.6

2.b.iii Algorithmes hybrides

Ce paragraphe s'intéresse aux algorithmes hybrides FP/DP, c'est à dire combinant deux ordonnancements : le premier à priorité fixe et le second à priorité dynamique. Ainsi, avec ce type d'algorithme, chaque tâche τ_i, activée à la date a, se voit attribuer les deux paramètres P_i et $P_i(t, a)$ définis à la sous section 2.a.iv. L'ordonnancement à priorités dynamiques est utilisé lorsque plusieurs tâches possèdent la même priorité fixe. Ainsi, à une date t, une tâche τ_i activée à la date a_i est strictement plus prioritaire qu'une tâche τ_j activée à la date a_j si $P_i < P_j$ ou si $P_i = P_j$ et $P_i(t, a_i) < P_j(t, a_j)$. Voici quelques exemples d'algorithmes FP/DP :

- FP/FIFO : Une implémentation classique de FP avec un ordonnancement FIFO lorsque plusieurs tâches possèdent la même priorité fixe.
- FP/RR : L'algorithme RR est utilisé pour ordonnancer les tâches de même priorité fixe.
- FP/LLF : Egalement appelé MUF (Maximum Urgency First), cette méthode est présentée par l'article [54].
- FP/EDF : L'algorithme EDF est utilisé pour ordonnancer les tâches de même priorité fixe. L'article [40] montre que, sous certaines conditions, FP/EDF domine FP/FIFO.

2.c Scénarios pire cas

Dans le cas de tâches périodiques, celles-ci s'exécutent indéfiniment. Malgré tout, leur caractère périodique limite la validation des contraintes temporelles à un intervalle de temps borné aussi appelé **intervalle de faisabilité**. Sachant qu'aucune faute temporelle ne peut être commise lorsque le processeur est libre, l'idée consiste à n'étudier que les intervalles de temps durant lesquels le processeur est utilisé en permanence. Ceci nous ramène à la notion de **période d'activité** du processeur déjà abordée à la sous section 2.a.v et détaillée à la sous section 2.c.i. Une fois la période d'activité, associée à une tâche, connue, celle-ci nous permet de déterminer les dates d'activation de cette tâche devant être testées afin d'en trouver le temps de réponse pire cas. Ainsi, la notion de période d'activité est primordiale dans les scénarios pires cas décrits à la sous section 2.c.ii.

2.c.i Périodes d'activité

Considérons un ensemble de n tâches noté τ. Au paragraphe 2.c.i.1, nous étudions la période d'activité la plus longue de cet ensemble. Puis, au paragraphe 2.c.i.2, nous introduisons la période d'activité de niveau de priorité P_i spécifique à l'ordonnancement FP. Enfin, au paragraphe 2.c.i.3, nous présentons, dans le cas d'un ordonnancement DP ou FP/DP, la période d'activité de niveau de priorité $PG_i(t_i)$.

2.c.i.1 Période d'activité la plus longue

Comme nous l'avons déjà dit, une période d'activité correspond à un intervalle de temps durant lequel le processeur est utilisé en permanence. Nous commençons donc par définir ce qu'est un ordonnanceur oisif :

- Un ordonnanceur est dit **oisif** lorsqu'il ne traîte pas nécessairement les tâches dès qu'elles sont activées alors qu'il n'a rien à faire. Cette caractéristique est principalement utilisée dans les réseaux afin de réguler le trafic.
- Un ordonnanceur est dit **non oisif** lorsqu'il ne peut retarder l'exécution d'une tâche s'il n'a rien à faire. Ce type d'ordonnanceur correspond au cas de la plupart des systèmes d'exploitation temps réel.

Les intervalles de temps durant lesquels le processeur est utilisé ou non sont identiques pour tous les algorithmes d'ordonnancement non oisifs. Nous introduisons maintenant la notion d'instant oisif puis la notion de période d'activité qui en découle :

Notion *Un instant oisif est un instant t tel que toutes les tâches activées avant la date t aient terminé leur exécution à cette date.*

Notion *Une période d'activité est un intervalle de temps $[a, b[$ tel que a et b sont deux instants oisifs et qu'il n'y a aucun instant oisif dans l'intervalle $]a, b[$.*

Comme l'expliquent les articles [30] et [38], la période d'activité du processeur la plus longue est observée lorsque toutes les tâches du système sont activées simultanément. Toutes les tâches étant activées simultanément, celles-ci ne peuvent être retardées par aucune autre tâche non préemptive. Ainsi, cette période d'activité est identique en contextes préemptif et non préemptif. Afin de déterminer la durée de cette période, nous devons calculer la durée cumulée des exécutions des tâches sur l'intervalle $[0, t[$, l'instant critique étant à la date 0. Sur cet intervalle, la tâche τ_i est activée $\lceil \frac{t}{T_i} \rceil$ fois, la durée cumulée des exécutions de τ_i vaut alors $\lceil \frac{t}{T_i} \rceil C_i$. Supposons maintenant que la première activation de l'une des tâches du système, disons τ_k, ait été retardée à la date $t_k > 0$, la durée cumulée des exécutions de τ_k sur l'intervalle $[0, t[$ vaudrait alors $\lceil \frac{\max(t-t_k,0)}{T_k} \rceil C_k$. Ainsi, le retard t_k sur la première activation de la tâche τ_k ne peut pas augmenter la durée cumulée de ses exécutions. Ceci prouve que la plus longue période d'activité du processeur est engendrée par un instant critique.

Notion *La fonction de travail $W(t)$ définit la durée cumulée des exécutions des n tâches de l'ensemble τ activées dans l'intervalle $[0, t[$:*

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

Cette fonction de travail est une fonction en escalier. Sachant que la droite d'équation $f(t) = t$ représente la capacité maximale de travail du processeur, celui-ci aura terminé l'exécution de toutes les activations survenues avant la date L lorsque $W(L) = f(L)$ soit $W(L) = L$. La durée L correspond alors à la plus longue période d'activité du système.

Propriété 2.1 *Soit L la durée de la plus longue période d'activité engendrée par les activations simultanées à la date 0 des n tâches de l'ensemble τ. Que le contexte soit préemptif ou non préemptif, L est solution de l'équation :*

$$L = \sum_{i=1}^{n} \lceil \frac{L}{T_i} \rceil C_i$$

Cette équation se résoud par la détermination du plus petit point fixe de la suite :

$$\begin{cases} L^0 = \sum_{i=1}^{n} C_i \\ L^{m+1} = \sum_{i=1}^{n} \lceil \frac{L^m}{T_i} \rceil C_i \end{cases}$$

Considérons, à titre d'exemple, le système composé des trois tâches dont les caractéristiques sont données dans le tableau 2.7. La charge de travail totale, pour ce système, est égale à $0,90$ et l'instant critique se répète périodiquement avec une période égale à 300.

Tâche	C	T
τ_1	10	30
τ_2	5	20
τ_3	8	25

TABLE 2.7 – Période d'activité : caractéristiques des tâches de l'exemple

Les différents points correspondant à la suite précédente sont symbolisés par les carrés sur la figure 2.15. Nous constatons alors que la plus longue période d'activité du processeur, pour ce système, vaut $L = 161$.

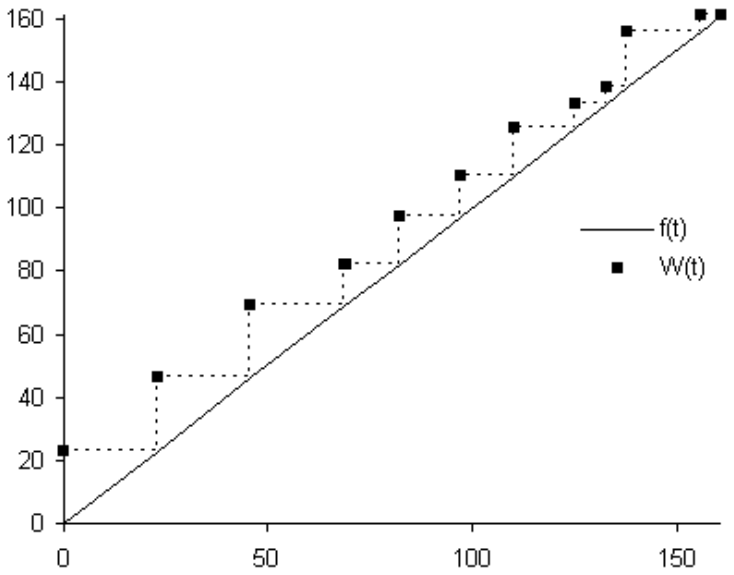

FIGURE 2.15 – Fonction de travail associée au système décrit dans le tableau 2.7

Si aucune faute temporelle ne se produit sur l'intervalle $[0, 161[$, alors aucune faute temporelle n'arrivera durant toute la vie de ce système. Nous remarquons qu'une fois cette période d'activité terminée le processeur ne restera pas inutilisé jusqu'au prochain instant critique. En effet, si tel était le cas, nous aurions une charge de travail totale de $161/300 = 0,54$. Or, notre système possède une charge de travail égale à $0,90$.

2.c.i.2 Période d'activité de niveau de priorité P_i

Dans le cas d'un ordonnancement FP, l'article [32] introduit la notion de période d'activité de niveau de priorité P_i où P_i correspond au paramètre, défini à la sous section 2.a.iv, inversement proportionnel à la priorité fixe d'une tâche τ_i. Avant de présenter cette période d'activité, nous introduisons la notion d'instant oisif de niveau de priorité P_i :

Notion *Un instant oisif de niveau de priorité P_i est un instant t tel que toute activation, survenue avant la date t, de n'importe quelle tâche τ_j de l'ensemble τ telle que $P_j \leq P_i$, ait terminé son exécution à la date t.*

Notion *Une période d'activité de niveau de priorité P_i est un intervalle de temps $[a, b[$ tel que a et b sont deux instants oisifs de niveau de priorité P_i et qu'il n'y a aucun instant oisif de niveau de priorité P_i dans l'intervalle $]a, b[$.*

Par rapport à la période d'activité la plus longue, étudiée au paragraphe 2.c.i.1, cette fois, nous ne considérons que les tâches appartenant à l'ensemble $\{\tau_j \in \tau / P_j \leq P_i\}$. Ainsi, si tant est qu'elle existe, une tâche non préemptive n'appartenant pas à cet ensemble peut retarder le début de cette période d'activité par effet non préemptif. Nous devons donc distinguer les contextes préemptif et non préemptif.

Soit L_i la durée de la plus longue période d'activité de niveau de priorité P_i, engendrée par les activations simultanées à la date 0 des tâches de l'ensemble $\{\tau_j \in \tau / P_j \leq P_i\}$:

Propriété 2.2 *En contexte préemptif, L_i correspond au plus petit point fixe de la suite :*

$$\begin{cases} L_i^0 = \sum\limits_{\tau_j \in \tau / P_j \leq P_i} C_j \\ L_i^{m+1} = \sum\limits_{\tau_j \in \tau / P_j \leq P_i} \lceil \frac{L_i^m}{T_j} \rceil C_j \end{cases}$$

Les thèses [23] et [39] ainsi que l'article [24] montrent que la notion de période d'activité de niveau de priorité P_i peut s'appliquer en contexte non préemptif à condition de tenir compte de l'effet non préemptif maximum obtenu par l'activation, juste avant le début de la période d'activité en question, de la tâche non préemptive de durée maximale appartenant à l'ensemble $\{\tau_k \in \tau / P_k > P_i\}$.

Propriété 2.3 *En contexte non préemptif, L_i correspond au plus petit point fixe de la suite :*

$$\begin{cases} L_i^0 = \sum\limits_{\tau_j \in \tau / P_j \leq P_i} C_j + \max^*_{\tau_k \in \tau / P_k > P_i}(C_k - 1) \\ L_i^{m+1} = \sum\limits_{\tau_j \in \tau / P_j \leq P_i} \lceil \frac{L_i^m}{T_j} \rceil C_j + \max^*_{\tau_k \in \tau / P_k > P_i}(C_k - 1) \end{cases}$$

Remarque *L'effet non préemptif ne pouvant être dû qu'à une tâche moins prioritaire que la tâche τ_i, si celle-ci est la moins prioritaire, cet effet ne peut avoir lieu. Dans ce cas, la durée L_i est identique en contextes préemptif et non préemptif.*

2.c.i.3 Période d'activité de niveau de priorité $PG_i(t_i)$

Dans ce paragraphe, nous ne nous intéressons qu'aux ordonnancements FP/DP. Toutefois, un ordonnancement DP étant assimilable à un ordonnancement FP/DP particulier où toutes les tâches possèdent la même priorité fixe, les notions et les formules présentées ici demeurent valables pour un ordonnancement DP.

Dans le cas d'un ordonnancement FP/DP, nous allons introduire la notion de période d'activité de niveau de priorité $PG_i(t_i)$ similaire à celle présentée au paragraphe 2.c.i.2 dans le cas d'un ordonnancement FP. Cependant, dans le cas présent, chaque tâche possède deux priorités : l'une fixe et l'autre dynamique. Nous allons donc étudier, dans un premier temps, la notion de priorité généralisée qui permet de comparer les priorités fixes et dynamiques de deux tâches différentes à l'aide d'un seul terme par tâche. Nous précisons que le paramètre $PG_i(t_i)$ représente la priorité généralisée de la tâche τ_i activée à la date t_i. Cette priorité généralisée n'est applicable qu'aux ordonnancements FP/DP dont la priorité dynamique est invariante. La notion de priorité invariante est définie par L.George dans l'article [22] :

Propriété 2.4 *La priorité d'une tâche τ_i activée à la date a est dite **invariante** si les propriétés suivantes sont vérifiées :*

- *La priorité d'une tâche activée à la date a demeure inchangée à toute date supérieure ou égale à a.*
- *La priorité d'une tâche activée à la date a_1 est supérieure à celle de la même tâche activée à la date a_2 si $a_2 > a_1$.*

Nous rappelons que le paramètre $P_i(t, a)$, associé à une tâche τ_i activée à la date a, est inversement proportionnel à sa priorité dynamique évaluée à la date t et vaut :

- Avec l'algorithme EDF, $\forall t \geq a$, $P_i(t, a) = a + D_i$.
- Avec l'algorithme LLF, $\forall t \geq a$, $P_i(t, a) = a + D_i - C_i(t) - t$ où $C_i(t)$ représente la durée pire cas de l'exécution restante à accomplir à la date t.
- Avec l'algorithme FIFO, $\forall t \geq a$, $P_i(t, a) = a$.

Ainsi, dans le cas d'un ordonnancement EDF, une fois la tâche τ_i activée à l'instant a_2, son paramètre $P_i(t, a_2)$ vaut $a_2 + D_i$ et demeure inchangé quelle que soit la date $t \geq a_2$ à laquelle celui-ci est évalué. Supposons maintenant que la tâche τ_i ait également été activée à l'instant $a_1 < a_2$, son paramètre $P_i(t, a_1)$ vaut alors $a_1 + D_i$ et vérifie, $\forall t \geq a_2 > a_1$, $P_i(t, a_1) < P_i(t, a_2)$. Ainsi, l'algorithme EDF satisfait bien la propriété 2.4 tout comme l'algorithme FIFO. La notion de priorité généralisée peut donc être employée, entre autres, avec les politiques EDF, FIFO, FP/EDF et FP/FIFO.

La notion de **priorité généralisée**, adaptée aux ordonnanceurs FP/DP dont la priorité dynamique vérifie la propriété 2.4, est présentée par L.George dans l'article [22]. La priorité généralisée de la tâche τ_i, activée à la date t_i, est inversement proportionnelle au paramètre $PG_i(t_i)$ défini comme suit :

$$PG_i(t_i) = \begin{cases} P_i \text{ si la priorité de la tâche } \tau_i \text{ est comparée à celle d'une} \\ \quad \text{autre tâche possédant une priorité fixe différente} \\ P_i(t_i, t_i) \text{ autrement} \end{cases}$$

Nous pouvons maintenant étendre les notions d'instant oisif et de période d'activité, exposées au paragraphe 2.c.i.2, à tout ordonnanceur de type FP/DP qui respecte la propriété 2.4 :

Notion *Un instant oisif de niveau de priorité $PG_i(t_i)$ est un instant t tel que toute activation, survenue à une date $t_j < t$, de n'importe quelle tâche τ_j de l'ensemble τ telle que $PG_j(t_j) \leq PG_i(t_i)$, ait terminé son exécution à la date t.*

Notion *Une période d'activité de niveau de priorité $PG_i(t_i)$ est un intervalle de temps $[a, b[$ tel que a et b sont deux instants oisifs de niveau de priorité $PG_i(t_i)$ et qu'il n'y a aucun instant oisif de niveau de priorité $PG_i(t_i)$ dans l'intervalle $]a, b[$.*

Nous cherchons, maintenant, à calculer la plus longue période d'activité de niveau de priorité $PG_i(t_i)$, notée $L_i(a)$, obtenue lorsque la tâche τ_i est activée pour la première fois à la date a telle que $a \in [0, T_i[$ et que toutes les tâches de l'ensemble $\{\tau_j \in \tau - \{\tau_i\} / PG_j(0) \leq PG_i(t_i)\}$ sont activées simultanément à la date 0. Bien évidement, la date t_i est de la forme $t_i = a + k \times T_i$ où $k \in N$. Nous précisons que, lors du calcul de $L_i(a)$, la date t_i est inconnue. C'est la connaissance de $L_i(a)$ qui nous permet de dire que la dernière date d'activation t_i de la tâche τ_i appartenant à l'intervalle $[0, L_i(a)[$ vaut $t_i = a + \lceil \frac{L_i(a)-a}{T_i} \rceil T_i$.

Propriété 2.5 *En contexte préemptif, $L_i(a)$ correspond au plus petit point fixe de la suite :*

$$\begin{cases} L_i(a)^0 = \sum\limits_{\tau_j \in \tau - \{\tau_i\} / PG_j(0) \leq PG_i(a)} C_j + \eta C_i \\ L_i(a)^{m+1} = \sum\limits_{\tau_j \in \tau - \{\tau_i\} / PG_j(0) \leq PG_i(L_i(a)^m)} \lceil \frac{L_i(a)^m}{T_j} \rceil C_j + \lceil \frac{\max(L_i(a)^m - a, 0)}{T_i} \rceil C_i \end{cases}$$

$$\text{où } \eta = \begin{cases} 1 \text{ si } a = 0 \\ 0 \text{ autrement} \end{cases}$$

Propriété 2.6 *En contexte non préemptif, $L_i(a)$ correspond au plus petit point fixe de la suite :*

$$\begin{cases} L_i(a)^0 = \sum\limits_{\tau_j \in \tau - \{\tau_i\} / PG_j(0) \leq PG_i(a)} C_j + \eta C_i + \max^*_{\tau_k \in \tau / PG_j(0) > PG_i(a)} (C_k - 1) \\ L_i(a)^{m+1} = \sum\limits_{\tau_j \in \tau - \{\tau_i\} / PG_j(0) \leq PG_i(L_i(a)^m)} \lceil \frac{L_i(a)^m}{T_j} \rceil C_j + \lceil \frac{\max(L_i(a)^m - a, 0)}{T_i} \rceil C_i + \\ \max^*_{\tau_k \in \tau / PG_j(0) > PG_i(L_i(a)^m)} (C_k - 1) \end{cases}$$

$$\text{où } \eta = \begin{cases} 1 \text{ si } a = 0 \\ 0 \text{ autrement} \end{cases}$$

2.c.ii Scénarios pires cas

A l'aide des différentes périodes d'activité étudiées à la sous section 2.c.i, nous allons maintenant établir les scénarios pires cas permettant la détermination du temps de réponse pire cas de chaque tâche de l'ensemble τ. Ces scénarios étant conditionnés par l'algorithme d'ordonnancement, nous commençons par examiner, au paragraphe 2.c.ii.1, les scénarios adaptés aux ordonnancements de type FP. Le paragraphe 2.c.ii.2 traite ensuite le cas des ordonnancements DP et FP/DP.

2.c.ii.1 Pour l'ordonnancement FP

L'ordonnancement FP n'étant pas hybride, nous rappelons que chacune des n tâches de l'ensemble τ possède une priorité fixe unique dans cet ensemble. Nous rappelons également que la priorité de la tâche τ_i est inversement proportionnelle à son paramètre constant P_i. Aussi, nous définissons les sous ensembles de τ suivants :

- L'ensemble des tâches strictement plus prioritaires que τ_i : $hp_i = \{\tau_j \in \tau / P_j < P_i\}$.
- L'ensemble des tâches strictement moins prioritaires que τ_i : $lp_i = \{\tau_j \in \tau / P_j > P_i\}$.

En contexte préemptif, le temps de réponse pire cas d'une tâche τ_i est obtenu dans la plus longue période d'activité de niveau de priorité P_i, notée L_i, lorsque toutes les tâches de l'ensemble $hp_i \cup \{\tau_i\}$ sont activées simultanément, au début de cette période d'activité, à la date 0. La figure 2.16 illustre ce scénario.

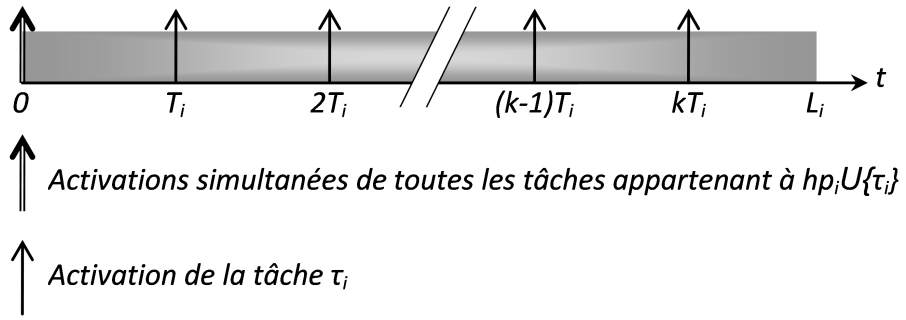

FIGURE 2.16 – Scénario pire cas pour une tâche τ_i avec FP préemptif

Sachant que, sur l'intervalle $[0, L_i[$, seules les tâches appartenant à $hp_i \cup \{\tau_i\}$ sont exécutées, à chacune de ses activations sur cet intervalle, la tâche τ_i est retardée soit par des tâches plus prioritaires qu'elle, soit du fait de l'une de ses activations précédentes. Ainsi, la date d'activation de la tâche τ_i menant à son temps de réponse pire cas appartient à l'ensemble $A_i = \{a_i^k = k \times T_i, k \in N / a_i^k \in [0, L_i - C_i[\}$. Notez que si la date $L_i - C_i$ coïncide avec une activation de la tâche τ_i celle-ci n'est pas étudiée. En effet, de part la définition de la période d'activité de niveau de priorité P_i, donnée au paragraphe 2.c.i.2, le temps de réponse de la tâche τ_i activée à la date $L_i - C_i$ serait obligatoirement égale à C_i ce qui correspond à son temps de réponse minimal. En conséquence, ce temps de réponse ne peut être supérieur à ceux observés sur les activations qui précèdent la date $L_i - C_i$.

En contexte non préemptif, le temps de réponse pire cas d'une tâche τ_i est également obtenu dans la plus longue période d'activité de niveau de priorité P_i lorsque toutes les tâches de l'ensemble $hp_i \cup \{\tau_i\}$ sont activées simultanément à la date 0. Dans ce cas, comme nous l'expliquons au paragraphe 2.c.i.2, si tant est qu'elle existe, nous devrons prendre en compte, lors du calcul de L_i, la tâche de durée maximale parmi celles moins prioritaires que τ_i.

2.c.ii.2 Pour l'ordonnancement DP ou FP/DP

Les scénarios pires cas mis en évidence dans ce sous paragraphe ne s'appliquent qu'aux algorithmes DP et FP/DP dont la priorité dynamique vérifie la propriété 2.4.

Supposons qu'une tâche τ_j activée à la date 0 soit plus prioritaire que la tâche τ_i activée à la date a, c'est à dire que nous ayons $PG_j(0) \leq PG_i(a)$. De part la propriété 2.4, nous savons que les activations suivantes de la tâche τ_j ne peuvent être que moins prioritaires que son activation à la date 0. Cependant, certaines d'entre elles peuvent tout de même s'avérer plus prioritaires que la tâche τ_i activée à la date a. Dans ce cas, nous disons que la tâche τ_j est potentiellement plus prioritaire que la tâche τ_i activée à la date a. A l'inverse, si nous avons $PG_j(0) > PG_i(a)$, assurément la tâche τ_j est toujours moins prioritaire que la tâche τ_i activée à la date a. Nous définissons maintenant les sous ensembles suivants :

- L'ensemble des tâches potentiellement plus prioritaires que la tâche τ_i activée à la date a autres que τ_i : $hp_i(a) = \{\tau_j \in \tau - \{\tau_i\} / PG_j(0) \leq PG_i(a)\}$.
- L'ensemble des tâches toujours moins prioritaires que la tâche τ_i activée à la date a : $lp_i(a) = \{\tau_j \in \tau / PG_j(0) > PG_i(a)\}$.

La propriété suivante, énoncée par L.George dans le papier [22], est à l'origine du scénario pire cas pour tout ordonnancement FP/DP. Sachant qu'un ordonnancement DP est assimilable à un cas particulier d'ordonnancement FP/DP où toutes les tâches possèdent la même priorité fixe, cette propriété est également valable pour les algorithmes DP.

Propriété 2.7 *Une tâche τ_i ordonnancée FP/DP, dont la première activation intervient à la date $a \in [0, T_i[$, obtient son temps de réponse maximum, dans la plus longue période d'activité de niveau de priorité $PG_i(t_i)$ obtenue lorsque toutes les tâches de l'ensemble $hp_i(t_i)$ sont activées simultanément à la date 0. Autrement dit, lorsque la première activation de la tâche τ_i survient à la date a, le temps de réponse maximum de cette tâche est obtenu par l'une de ses activations sur l'intervalle $[0, L_i(a)[$.*

De la propriété précédente, nous déduisons que, lorsqu'une tâche τ_i est activée pour la première fois à la date a, son temps de réponse maximum est obtenue suite à son activation à l'une des dates de l'ensemble $A_i(a)$ lorsque toutes les tâches de l'ensemble $hp_i(L_i(a))$ sont activées simultanément à la date 0. L'ensemble $A_i(a)$ vaut :

$$A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$$

Au premier abord, nous pourrions penser que, pour trouver le temps de réponse pire cas d'une tâche τ_i, comme dans le cas d'un ordonnancement FP, nous pouvons nous contenter du cas où sa première activation arrive à la date 0. Autrement dit, nous pourrions croire que l'obtention de son temps de réponse pire cas ne soit basée que sur l'ensemble $A_i(0)$. La figure 2.17, qui illustre l'ordonnancement EDF des deux tâches préemptives décrites au tableau 2.8, montre que cette idée est fausse.

Tâche	*C*	*T*	*D*
τ_1	1	6	4
τ_2	3	7	5

TABLE 2.8 – Caractéristiques des tâches illustrées à la figure 2.17

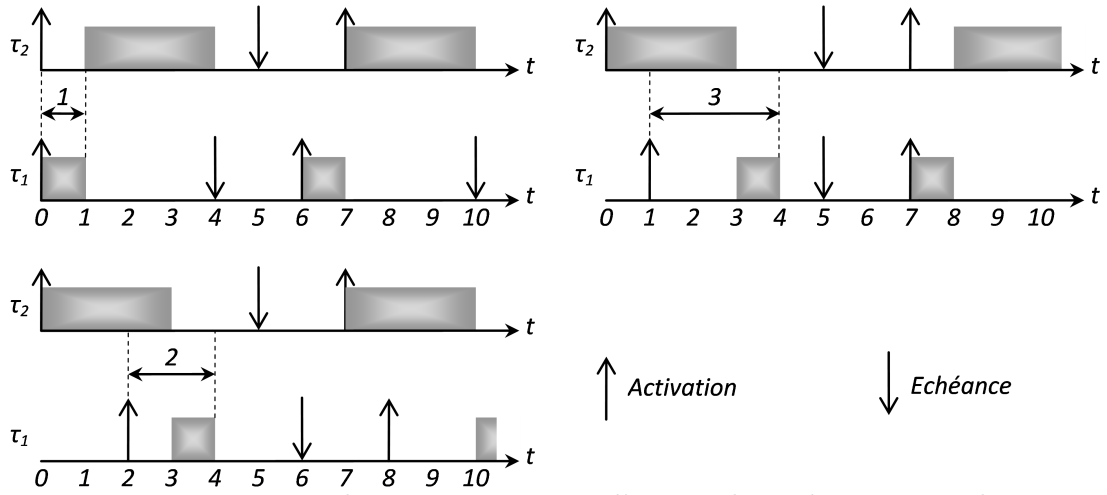

FIGURE 2.17 – Temps de réponse pire cas d'une tâche ordonnancée selon EDF

Soit a la date de la première activation de la tâche τ_1. A la figure 2.17, nous observons trois démarrages possibles selon que la date a soit égale à 0, 1 ou 2. Dans tous les cas, la tâche τ_2 est toujours activée à la date 0. Nous remarquons que le temps de réponse pire cas de la tâche τ_1 vaut 3 et est obtenu lorsque la date a vaut 1 et non 0. Bien entendu, nous avons vérifié les cas où a est compris entre 3 et 5. Ceux-ci, n'apportant pas de temps de réponse supérieur à 3, n'ont pas été ajoutés à la figure 2.17. Nous soulignons qu'étudier un scénario avec a supérieur ou égal à T_1 serait dépourvu d'intérêt puisque cette date d'activation est déjà analysée dans le cas où la première activation de la tâche τ_1 survient à la date a *modulo* T_1.

L'observation précédente tient au fait que, dans le cas d'un ordonnancement DP ou FP/DP, la priorité d'une tâche dépendant de sa date d'activation, entre deux activations différentes, certaines tâches qui étaient moins prioritaires pour l'une peuvent devenir plus prioritaires pour l'autre. C'est pourquoi, l'obtention du temps de réponse pire cas d'une tâche τ_i nécessite l'étude de chacune de ses dates de première activation appartenant à l'intervalle $[0, T_i[$. Finalement, le calcul du temps de réponse pire cas d'une tâche τ_i nécessite l'étude de chaque date d'activation appartenant à A_i :

$$A_i = \bigcup_{a \in [0,T_i[} A_i(a) \text{ avec } A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$$

Selon que le contexte soit préemptif ou non, le calcul de $L_i(a)$ suit la propriété 2.5 ou 2.6.

2.d Conditions de faisabilité FP/FIFO

Les scénarios pires cas présentés à la sous section 2.c.ii vont maintenant nous permettre de calculer le temps de réponse pire cas, noté r_i, de chaque tâche τ_i d'un ensemble de n tâches noté τ. Avec le facteur d'utilisation du processeur, noté U, ces temps de réponse pires cas apportent une condition nécessaire et suffisante de faisabilité qui consiste à vérifier que $\forall \tau_i \in \tau, r_i \leq D_i$ et $U \leq 1$. Cette condition nécessaire et suffisante est valable pour les contextes préemptif et non préemptif. Les équations permettant la détermination des temps de réponse pires cas avec les ordonnancements FP, FIFO et FP/FIFO sont respectivement exposées aux sous sections 2.d.i, 2.d.ii et 2.d.iii.

2.d.i Ordonnancement FP

En contexte préemptif, comme l'expliquent les papiers [32] et [55], ainsi que le scénario pire cas décrit au paragraphe 2.c.ii.1, le temps de réponse pire cas de chaque tâche τ_i correspond à son temps de réponse maximum observé sur l'ensemble de ses dates d'activations $A_i = \{a_i^k = k \times T_i, k \in N / a_i^k \in [0, L_i - C_i[\}$. Nous rappelons que le calcul de la plus longue période d'activité de niveau de priorité P_i, notée L_i, est détaillé à la propriété 2.2.

Supposons que la tâche τ_i ait été activée à la date t, nous souhaitons calculer la date $\omega_{i,t}$ à laquelle l'exécution de celle-ci s'achève. Concernant la tâche τ_i, lorsque celle-ci est activée à la date t, son exécution ne peut débuter qu'une fois ses activations précédentes accomplies. Ainsi, nous devons comptabiliser l'exécution de chacune des activations de la tâche τ_i sur l'intervalle $[0, t]$ ce qui nous mène au terme suivant :

$$(1 + \lfloor \tfrac{t}{T_i} \rfloor) C_i$$

Tant que l'exécution de la tâche τ_i, activée à la date t, n'est pas terminée, les tâches, plus prioritaires que celle-ci, la préemptent à chacune de leurs activations et s'exécutent. Ainsi, les tâches plus prioritaires que τ_i s'exécutent normalement sur l'intervalle $[0, \omega_{i,t}[$. Notez que la date $\omega_{i,t}$ est exclue de l'intervalle précédent car à cette date l'exécution de la tâche τ_i activée à la date t est terminée. Par conséquent, les activations éventuelles de tâches plus prioritaires que τ_i à cette date doivent être ignorées. La prise en compte des tâches plus prioritaires que τ_i revient donc à :

$$\sum_{\tau_j \in hp_i} \lceil \tfrac{\omega_{i,t}^n}{T_j} \rceil C_j$$

Les deux termes précédents expliquent le théorème suivant :

Théorème 2.1 *Le temps de réponse pire cas d'une tâche τ_i ordonnancée FP, en contexte préemptif, est solution de $r_i = \max_{t \in A_i}(\omega_{i,t} - t)$ où $\omega_{i,t}$ est la date de fin d'exécution de la tâche τ_i activée à la date t et solutionne l'équation suivante :*

$$\omega_{i,t} = (1 + \lfloor \tfrac{t}{T_i} \rfloor) C_i + \sum_{\tau_j \in hp_i} \lceil \tfrac{\omega_{i,t}}{T_j} \rceil C_j$$

Avec $A_i = \{a_i^k = k \times T_i, k \in N / a_i^k \in [0, L_i - C_i[\}$

Dans le cas particulier où une tâche τ_i appartenant à τ vérifierait $D_i \leq T_i$, la fin de son exécution, lorsque celle-ci est activée à la date 0, coïnciderait avec la fin de la plus longue période d'activité de niveau de priorité P_i. Ainsi, nous aurions $L_i \leq T_i$ et, en conséquence, $A_i = \{0\}$. Finalement, dans ces conditions, seule l'activation de la tâche τ_i à la date 0 doit être testée et son temps de réponse pire cas est alors, comme expliqué dans l'article [29], solution de l'équation suivante :

$$r_i = \omega_{i,0} \text{ où } \omega_{i,0} \text{ est solution de } \omega_{i,0} = C_i + \sum_{\tau_j \in hp_i} \lceil \tfrac{\omega_{i,0}}{T_j} \rceil C_j$$

En contexte non préemptif, le temps de réponse pire cas d'une tâche τ_i de priorité P_i pourrait être déterminé à l'aide du théorème 2.1 en tenant compte de l'effet non préemptif dû à la tâche de durée maximale parmi les tâches moins prioritaires que celle-ci. Le résultat ne serait alors qu'une condition suffisante mais non nécessaire du fait que des tâches plus prioritaires seraient prises en compte pendant l'exécution de la tâche τ_i alors que celle-ci est non préemptive. C'est pourquoi, nous nous intéresserons à la condition nécessaire et suffisante exposée dans la thèse [23] et dans le rapport [24]. Cette condition s'appuie sur le calcul, pour chaque date d'activation de la tâche τ_i appartenant à l'ensemble $A_i = \{a_i^k = k \times T_i, k \in N / a_i^k \in [0, L_i - C_i[\}$, de la date à laquelle débute l'exécution correspondante. Nous rappelons que le calcul, en contexte non préemptif, de la plus longue période d'activité de niveau de priorité P_i, notée L_i, est détaillé à la propriété 2.3.

Supposons que la tâche τ_i ait été activée à la date t, nous souhaitons calculer la date $\overline{\omega}_{i,t}$ après laquelle l'exécution de celle-ci débute immédiatement. Naturellement cette exécution n'entre pas en considération lors du calcul de la date $\overline{\omega}_{i,t}$. Par contre, les activations de la tâche τ_i survenues avant la date t doivent être prises en compte. Ainsi, nous devons comptabiliser l'exécution de chacune des activations de la tâche τ_i sur l'intervalle $[0, t[$ sachant que son activation à la date t sera ajoutée ultérieurement une fois la date $\overline{\omega}_{i,t}$ connue. Ceci nous mène au terme suivant :

$$((\lfloor \frac{t}{T_i} \rfloor + 1) - 1)C_i = \lfloor \frac{t}{T_i} \rfloor C_i$$

Les tâches plus prioritaires que τ_i s'exécutent normalement à chacune de leurs activations sur l'intervalle $[0, \overline{\omega}_{i,t}]$. Notez que la date $\overline{\omega}_{i,t}$ est inclue dans l'intervalle précédent car c'est immédiatement après cette date que la tâche τ_i activée à la date t débute son exécution. Ainsi, toute tâche plus prioritaire activée avant le début de cette exécution la retarde. Nous aboutissons alors au terme :

$$\sum_{\tau_j \in hp_i} (1 + \lfloor \frac{\overline{\omega}_{i,t}^n}{T_j} \rfloor) C_j$$

Les deux termes précédents ajoutés à l'effet non préemptif expliquent le théorème suivant :

Théorème 2.2 *Le temps de réponse pire cas d'une tâche τ_i ordonnancée FP, en contexte non préemptif, est solution de $r_i = \max_{t \in A_i}(\overline{\omega}_{i,t} + C_i - t)$ où $\overline{\omega}_{i,t}$ est la date de début d'exécution de la tâche τ_i activée à la date t et solutionne l'équation suivante :*

$$\overline{\omega}_{i,t} = \lfloor \frac{t}{T_i} \rfloor C_i + \sum_{\tau_j \in hp_i} (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_j} \rfloor) C_j + \max^*_{\tau_j \in lp_i}(C_k - 1)$$

Avec $A_i = \{a_i^k = k \times T_i, k \in N / a_i^k \in [0, L_i - C_i[\}$

2.d.ii Ordonnancement FIFO

Avec l'ordonnancement FIFO, toutes les tâches sont exécutées dans l'ordre chronologique de leurs activations. Ainsi, la tâche en cours d'exécution correspond toujours à la plus ancienne tâche activée. De plus, aucune tentative de préemption n'étant possible, la nature préemptive ou non de chaque tâche n'intervient pas dans le calcul des temps de réponse pires cas. Nous rappelons que cette politique d'ordonnancement revient à attribuer aux tâches des priorités dynamiques correspondant à leurs dates d'activation. Toute tâche activée au plus tard à la date t est donc plus prioritaire que la tâche τ_i activée à cette même date.

Comme nous l'avons fait, en contexte préemptif, avec la politique d'ordonnancement FP, à la sous section 2.d.i, nous devons déterminer la date de fin d'exécution, notée $\omega_{i,t}$, de la tâche τ_i activée à la date t. Conformément au scénario pire cas décrit au paragraphe 2.c.ii.2, la date d'activation t vérifie $t = a+k\times T_i$ où $a \in [0, T_i[$ et $k \in N$. En effet, alors que toutes les autres tâches sont activées simultanément à la date 0, la première activation de la tâche τ_i survient à la date a. Comme nous venons de le rappeler, toute tâche activée au plus tard à la date t, c'est à dire sur l'intervalle $[0, t]$, s'avère plus prioritaire que la tâche τ_i activée à cette même date et en retarde l'exécution. Nous devons donc comptabiliser, durant le calcul de la date $\omega_{i,t}$ les exécutions de toutes les tâches, autres que τ_i, activées sur l'intervalle $[0, t]$:

$$\sum_{\tau_j \in \tau - \{\tau_i\}} (1 + \lfloor \tfrac{t}{T_j} \rfloor) C_j$$

Concernant la tâche τ_i, nous devons considérer toutes ses activations survenues sur l'intervalle $[a, t]$:

$$(1 + \lfloor \tfrac{\max(t-a,0)}{T_i} \rfloor) C_i$$

Sachant que $t = a+k\times T_i$ avec $k \in N$ et $a \in [0, T_i[$, le terme précédent se simplifie comme suit :

$$(1 + \lfloor \tfrac{\max(t-a,0)}{T_i} \rfloor) C_i = (1 + \lfloor \tfrac{t}{T_i} \rfloor) C_i$$

Aucun des termes précédents ne dépendant de la date $\omega_{i,t}$, nous aboutissons au théorème :

Théorème 2.3 *Le temps de réponse pire cas d'une tâche τ_i ordonnancée FIFO est solution de l'équation suivante :*

$$r_i = \max_{t \in A_i}(\omega_{i,t} - t) \ \textit{avec}\ \omega_{i,t} = \sum_{\tau_j \in \tau} (1 + \lfloor \tfrac{t}{T_j} \rfloor) C_j$$

Avec $A_i = \bigcup_{a \in [0, T_i[} A_i(a)$ *et* $A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$.

Nous rappelons que le calcul de la période d'activité $L_i(a)$ est expliqué par la propriété 2.5.

2.d.iii Ordonnancement FP/FIFO

A l'aide des théorèmes 2.1 2.2 et 2.3, nous pouvons maintenant déterminer les temps de réponse pires cas pour l'ordonnancement FP/FIFO en contextes préemptif et non préemptif. Comme l'explique la thèse [39], la politique FP/FIFO permet d'ordonnancer des systèmes non ordonnançables par la politique FP. Au sein d'un ensemble de n tâches noté τ, plusieurs tâches peuvent posséder la même priorité fixe. Aussi, nous définissons le sous ensemble des tâches de même priorité que la tâche τ_i appartenant à τ sauf τ_i :

$$sp_i = \{\tau_j \in \tau - \{\tau_i\} / P_j = P_i\}$$

En contexte préemptif, comme nous l'avons fait pour la politique d'ordonnancement FP, à la sous section 2.d.i, nous calculons la date de fin d'exécution $\omega_{i,t}$ de la tâche τ_i activée à la date t. Sachant que les tâches de l'ensemble $sp_i \cup \{\tau_i\}$ possèdent toutes la même priorité fixe, selon le raisonnement mené à la sous section 2.d.ii, leur contribution s'élève à :

$$\sum_{\tau_j \in sp_i \cup \{\tau_i\}} (1 + \lfloor \tfrac{t}{T_j} \rfloor) C_j$$

Comme dans le cas de la politique FP, en contexte préemptif, les tâches dotées d'une priorité fixe plus forte peuvent préempter la tâche en cours d'exécution pour s'exécuter sur l'intervalle $[0, \omega_{i,t}[$:

$$\sum_{\tau_j \in hp_i} \lceil \tfrac{\omega_{i,t}}{T_j} \rceil C_j$$

Les deux termes précédents nous mènent au théorème suivant :

Théorème 2.4 *Le temps de réponse pire cas d'une tâche τ_i ordonnancée FP/FIFO, en contexte préemptif, est égal à $r_i = \max_{t \in A_i}(\omega_{i,t} - t)$ où $\omega_{i,t}$ correspond à la date de fin d'exécution de la tâche τ_i activée à la date t et solutionne l'équation suivante :*

$$\omega_{i,t} = \sum_{\tau_j \in sp_i \cup \{\tau_i\}} (1 + \lfloor \tfrac{t}{T_j} \rfloor) C_j + \sum_{\tau_j \in hp_i} \lceil \tfrac{\omega_{i,t}}{T_j} \rceil C_j$$

Avec $A_i = \bigcup_{a \in [0, T_i[} A_i(a)$ *et* $A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$.

En contexte non préemptif, nous devons calculer la date de début d'exécution $\overline{\omega}_{i,t}$ de la tâche τ_i activée à la date t. Lors de ce calcul, nous ne comptabilisons pas l'exécution liée à l'activation de la tâche τ_i à la date t qui sera prise en compte une fois la date $\overline{\omega}_{i,t}$ connue. Le terme correspondant aux tâches de l'ensemble $sp_i \cup \{\tau_i\}$ devient alors :

$$\sum_{\tau_j \in sp_i} (1 + \lfloor \tfrac{t}{T_j} \rfloor) C_j + \lfloor \tfrac{t}{T_i} \rfloor C_i$$

Nous intégrons maintenant les exécutions des tâches plus prioritaires activées sur l'intervalle $[0, \overline{\omega}_{i,t}]$. Nous rappelons que l'intervalle est fermé car une tâche plus prioritaire que la tâche τ_i peut en retarder l'exécution tant que celle-ci n'a pas commencé :

$$\sum_{\tau_j \in hp_i} (1 + \lfloor \tfrac{\overline{\omega}_{i,t}}{T_j} \rfloor) C_j$$

Les deux termes précédents ajoutés à l'effet non préemptif nous mènent directement au théorème suivant :

Théorème 2.5 *Le temps de réponse pire cas d'une tâche τ_i ordonnancée FP/FIFO, en contexte non préemptif, est égal à $r_i = \max_{t \in A_i}(\overline{\omega}_{i,t} + C_i - t)$ où $\overline{\omega}_{i,t}$ correspond à la date de début d'exécution de la tâche τ_i activée à la date t et solutionne l'équation suivante :*

$$\overline{\omega}_{i,t} = \sum_{\tau_j \in hp_i} (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_j} \rfloor) C_j + \sum_{\tau_j \in sp_i(t)} (1 + \lfloor \frac{t}{T_j} \rfloor) C_j + \lfloor \frac{t}{T_i} \rfloor C_i + \max^*_{\tau_k \in lp_i}(C_k - 1)$$

Avec $A_i = \bigcup_{a \in [0, T_i[} A_i(a)$ *et* $A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$.

2.e Conditions de faisabilité EDF

Comme nous l'avons vu au paragraphe 2.b.ii.1, le facteur d'utilisation peut, dans certains cas, se révéler nécessaire et suffisant à la validation d'un ensemble de tâches ordonnancées selon l'algorithme EDF. Dans le cas général, deux méthodes peuvent être employées pour vérifier la faisabilité d'un ensemble de tâches périodiques ou sporadiques ordonnancées avec cette politique. La première méthode, présentée à la sous section 2.e.i, consiste à étudier la demande processeur. La seconde, détaillée à la sous section 2.e.ii, s'appuie sur l'analyse des temps de réponse pires cas et autorise ainsi un diagnostic précis.

2.e.i Conditions de faisabilité d'EDF basées sur la demande processeur

Considérons un ensemble de tâches périodiques ou sporadiques τ. En contexte préemptif une condition nécessaire et suffisante de faisabilité, basée sur $h(t)$, est proposée dans le papier [7]. Cette condition consiste à vérifier, dans le scénario d'activation synchrone exposé au paragraphe 2.c.ii.2 que $\forall t \in [0, L[, h(t) \leq t$:

Théorème 2.6 *Une condition de faisabilité, nécessaire et suffisante, pour EDF en contexte préemptif est : $\forall t \in [0, L[, h(t) \leq t$ où $h(t) = \sum_{\tau_i \in \tau / D_i \leq t} \Big(1 + \lfloor \frac{t - D_i}{T_i} \rfloor\Big) C_i$.*

Notez que si $\forall \tau_i \in \tau, D_i \geq T_i$, l'article [7] montre que le théorème précédent équivaut à la condition $U \leq 1$. Dans ce cas, la condition $U \leq 1$ apparaît donc comme une condition de faisabilité nécessaire et suffisante. Nous précisons que l'article [38] démontre ce résultat dans le cas où $\forall \tau_i \in \tau, D_i = T_i$.

Dans sa thèse [23] ainsi que dans l'article [24], L.George montre que le test précédent peut être étendu au contexte non préemptif en ajoutant l'effet non préemptif dû à, si tant est qu'elle existe, la tâche possédant la durée d'exécution pire cas la plus grande parmi l'ensemble des tâches moins prioritaires au sens EDF que celles prises en compte par la demande processeur $h(t)$.

Théorème 2.7 *Une condition de faisabilité, nécessaire et suffisante, pour EDF en contexte non préemptif est : $\forall t \in [0, L[, h(t) + \max_{\tau_k \in \tau / D_k > t}(C_k - 1) \leq t$ où $h(t) = \sum_{\tau_i \in \tau / D_i \leq t} \Big(1 + \lfloor \frac{t - D_i}{T_i} \rfloor\Big) C_i$.*

2.e.ii Conditions de faisabilité d'EDF basées sur l'analyse des temps de réponse pires cas

Nous abordons maintenant les conditions de faisabilité basées sur l'analyse des temps de réponse pires cas. Soit une tâche τ_i, nous notons son temps de réponse pire cas r_i. Un ensemble de tâches périodiques ou sporadiques τ est alors ordonnançable avec la politique EDF si, et seulement si, $\forall \tau_i \in \tau, r_i \leq D_i$ et $U \leq 1$. Le paragraphe 2.e.ii.1 expose le calcul des temps de réponse pires cas en contexte préemptif. Le contexte non préemptif est ensuite traité au paragraphe 2.e.ii.2.

2.e.ii.1 Calcul des temps de réponse pires cas en contexte préemptif

Soit une tâche τ_i, appartenant à l'ensemble de tâches préemptives τ, activée à la date t_0. Comme expliqué au paragraphe 2.c.i.3, la priorité dynamique de la tâche τ_i, à la date $t_1 \geq t_0$, est inversement proportionnelle au paramètre $P_i(t_1, t_0)$ qui vaut alors $t_0 + D_i$. Nous calculons maintenant son temps de réponse pire cas, noté r_i, en nous basant sur le scénario pire cas décrit au paragraphe 2.c.ii.2.

Nous calculons la date $\omega_{i,t}$ à laquelle se termine l'exécution de la tâche τ_i, activée à la date $t = a + k \times T_i$ où $k \in N$. Nous rappelons que la date a correspond à la première activation de la tâche τ_i et vérifie $a \in [0, T_i[$. Nous soulignons que la date a est égale à $s_i(t) = t - \lfloor \frac{t}{T_i} \rfloor$. Les tâches qui interviennent dans ce calcul sont les suivantes :

- Soit τ_j une tâche de $hp_i(t)$, celle-ci possède $1 + \lfloor \frac{t+D_i-D_j}{T_j} \rfloor$ échéances inférieures ou égales à $t + D_i$. Ajoutons que $\lceil \frac{\omega_{i,t}}{T_j} \rceil$ activations de la tâche τ_j interviennent dans l'intervalle $[0, \omega_{i,t}[$. Ainsi, durant le calcul de la date $\omega_{i,t}$, nous considérerons $\min(1 + \lfloor \frac{t+D_i-D_j}{T_j} \rfloor, \lceil \frac{\omega_{i,t}}{T_j} \rceil)$ exécutions de chaque tâche τ_j de l'ensemble $hp_i(t)$.
- Concernant la tâche τ_i, seules ses activations sur l'intervalle $[s_i(t), t]$ possèdent une échéance inférieure ou égale à $t+D_i$. De plus, nous ne considérons que les activations ayant lieu sur l'intervalle $[s_i(t), \omega_{i,t}[$. Durant le calcul de la date $\omega_{i,t}$, nous comptons donc $\min(1 + \lfloor \frac{t}{T_i} \rfloor, \lceil \frac{\max(\omega_{i,t} - s_i(t), 0)}{T_i} \rceil)$ exécutions de la tâche τ_i.

Le théorème suivant, concernant la détermination du temps de réponse pire cas d'une tâche préemptive τ_i appartenant à un ensemble de n tâches noté τ ordonnancé selon la politique EDF, est expliqué par M.Spuri dans l'article [52] :

Théorème 2.8 *Le temps de réponse pire cas d'une tâche τ_i, ordonnancée selon la politique EDF, en contexte préemptif, est donné par : $r_i = \max_{t \in A_i}(\omega_{i,t} - t)$ où :*

$$\omega_{i,t} = \sum_{\tau_j \in hp_i(t)} \min(1 + \lfloor \frac{t+D_i-D_j}{T_j} \rfloor, \lceil \frac{\omega_{i,t}}{T_j} \rceil) C_j + \min(1 + \lfloor \frac{t}{T_i} \rfloor, \lceil \frac{\max(\omega_{i,t} - s_i(t), 0)}{T_i} \rceil) C_i$$

Avec $A_i = \bigcup_{a \in [0, T_i[} A_i(a)$, $A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$ *et* $s_i(t) = t - \lfloor \frac{t}{T_i} \rfloor$

2.e.ii.2 Calcul des temps de réponse pires cas en contexte non préemptif

En contexte non préemptif, l'équation nécessaire au calcul du temps de réponse pire cas d'une tâche τ_i, activée à la date t, est semblable à celle présentée au théorème 2.8 mais tient compte de l'effet non préemptif dû aux tâches de l'ensemble $lp_i(t)$. La tâche τ_i étant non préemptive, nous calculons, cette fois, la date, notée $\overline{\omega}_{i,t}$, après laquelle son exécution débute immédiatement. Tant que la tâche τ_i n'a pas entamé son exécution, les tâches plus prioritaires peuvent s'exécuter. Ainsi, nous intégrons l'exécution des tâches plus prioritaires dans l'intervalle $[0, \overline{\omega}_{i,t}]$. Bien évidement, le calcul de la date $\overline{\omega}_{i,t}$ ne tient pas compte de l'exécution de la tâche τ_i activée à la date t. Cette dernière est ajoutée à la date $\overline{\omega}_{i,t}$ une fois celle-ci connue.

Théorème 2.9 *Le temps de réponse pire cas d'une tâche τ_i, ordonnancée selon la politique EDF, en contexte non préemptif, est donné par : $r_i = \max_{t \in A_i}(\overline{\omega}_{i,t} + C_i - t)$ où :*

$$\overline{\omega}_{i,t} = \sum_{\tau_j \in hp_i(t)} (1 + \lfloor \frac{\min(t + D_i - D_j, \overline{\omega}_{i,t})}{T_j} \rfloor) C_j + \min(\lfloor \frac{t}{T_i} \rfloor, 1 + \lfloor \frac{\max(\overline{\omega}_{i,t} - s_i(t), 0)}{T_i} \rfloor) C_i + \max_{\tau_k \in lp_i(t)} (C_k - 1)$$

Avec $A_i = \bigcup_{a \in [0, T_i[} A_i(a)$, $A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$ *et* $s_i(t) = t - \lfloor \frac{t}{T_i} \rfloor$

2.f Synthèse

Ce chapitre rappelle les notions essentielles à toute analyse temps réel telles que les modèles temporels des tâches, la notion de contrainte temporelle, les différents algorithmes d'ordonnancement ainsi que les scénarios pires cas et les conditions de faisabilité associées à chacun de ces algorithmes. Dans le cas d'un ordonnancement FP/FIFO, les conditions de faisabilité classiques sont particulièrement détaillées en vue du chapitre 4 où nous les améliorons en intégrant le coût de l'exécution d'un système d'exploitation temps réel conforme au standard OSEK. Bien que ce standard prescrive la politique à priorités fixes FP/FIFO, comme nous l'avons vu à travers ce chapitre, la politique à priorités dynamiques EDF s'avère plus optimale. Ainsi, cette politique nous a permis d'accroître le potentiel de notre exécutif OSEK. De même que pour celles associées à la politique FP/FIFO, les conditions de faisabilité de l'algorithme d'ordonnancement EDF sont reprises et améliorées au chapitre 5.

Chapitre 3

Présentation d'OSEK

Abréviation de "Offene Systeme und deren Schnittstellen für die Electronik im Kraftfahrzeug", traduisez "systèmes ouverts et leurs interfaces pour l'électronique automobile", le standard OSEK a vu le jour en 1993 sous l'impulsion de plusieurs industriels automobiles dont BMW, Bosch, DaimlerChrysler, Opel, Siemens, et Volkswagen, ainsi qu'un département de l'Université de Karlsruhe. Comme tout système d'exploitation conforme aux besoins de l'industrie automobile, OSEK requiert peu de ressources en termes de mémoires, vive et morte, et de puissance de calcul. Disponible sur une large gamme de processeurs 8, 16 et 32 bits, l'implémentation modulaire de ce système en autorise l'ajustement aux besoins réels de l'application. De plus, ce système d'exploitation temps réel est statique. Autrement dit, tous les objets du système, comme les tâches et les ressources, sont définis à la compilation. Aucune création ou destruction dynamique d'objets n'est possible. La section 3.a décrit les différentes politiques d'ordonnancement admises par la norme OSEK. La gestion des tâches est ensuite expliquée à la section 3.b. La méthode du plafond de priorité, prescrite par ce standard pour gérer l'accès aux ressources, est détaillée à la section 3.c. Nous verrons également comment cette méthode élimine tout risque d'interblocage entre différentes tâches. Le mécanisme des alarmes, sur lequel nous nous basons pour créer les tâches périodiques, est présenté à la section 3.d. Bien évidemment, la gestion des tâches ainsi que celle des alarmes et du mécanisme du plafond de priorité requièrent l'utilisation du processeur. Les différentes charges, dues à l'exécution d'un noyau OSEK, sont identifiées et caractérisées à la section 3.e. Enfin, la section 3.f résume les principales caractéristiques de ce standard.

3.a Politiques d'ordonnancement

La norme OSEK se base sur un ordonnanceur guidé par les événements. Autrement dit, l'ordonnanceur est invoqué à chaque fois qu'un événement, comme l'activation ou la terminaison d'une tâche, survient. Tout système d'exploitation OSEK étant statique, celui-ci n'utilise naturellement que des priorités fixes. Ainsi, les priorités affectées aux tâches qui constituent l'application demeurent invariantes durant toute sa vie. **Toute tâche τ_i se voit donc attribuer un paramètre P_i. Contrairement à la notation définie à la sous section 2.a.iv, selon le standard OSEK, le paramètre P_i est proportionnel à la priorité de la tâche τ_i. La priorité la plus haute n'étant pas précisée par ce standard, celle-ci dépend directement de l'implémentation. La priorité la plus basse, quant à elle, est associée au paramètre nul.** Par ailleurs, la norme OSEK définit trois types de traitement : les interruptions, le système d'exploitation et les tâches. La figure 3.1 montre que le système d'exploitation est plus prioritaire que n'importe quelle tâche applicative mais moins prioritaire que les interruptions. Au sein des interruptions, deux catégories, nommées ISR1 et ISR2, se distinguent. Quelque soit son type, une interruption est toujours déclenchée de façon événementielle : par exemple, lorsqu'un message arrive sur un bus de communication, ou encore lorsqu'un événement extérieur se produit tel que le blocage d'une roue de voiture. L'intérêt d'une interruption réside en sa capacité à interrompre le programme en cours d'exécution pour traiter immédiatement l'événement qui l'a déclenchée : en récupérant le message avant que celui-ci ne soit perdu, ou bien en déclenchant l'ABS. La différence entre les deux types précédents tient au fait que les interruptions de type ISR2 peuvent appeler certains services du système d'exploitation et sont vues par celui-ci. Au contraire, les interruptions de type ISR1 demeurent invisibles pour le système d'exploitation et ne peuvent en aucun cas en solliciter les services.

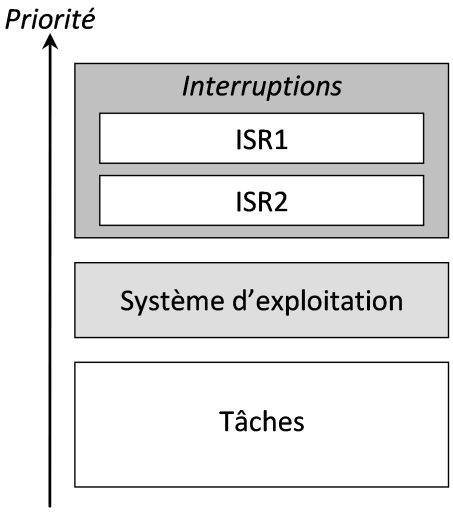

FIGURE 3.1 – Niveaux de priorités associés aux différents traitements

OSEK met en oeuvre un ordonnancement de type FP/FIFO, déjà présenté à la sous section 2.b.iii. Par ailleurs, les trois contextes préemptif, non préemptif et mixte, illustrés au paragraphe 2.a.iii.2, sont admis.

3.b Gestion des tâches

Rappelons que les systèmes multitâches ont été créés, entre autres, pour éviter qu'un processeur ne soit mobilisé par une tâche en attente d'un événement tel que le changement d'état d'une entrée, ou encore la réception d'un message. Bien qu'un tel système permette d'exécuter plusieurs tâches simultanément, une architecture monoprocesseur ne peut, quant à elle, en exécuter qu'une seule à instant donné. En conséquence, afin de concilier ces deux objectifs, durant l'exécution de l'application, à tout instant, chacune de ses tâches emprunte obligatoirement l'un des trois états suivants :

- **Courant** : C'est l'état de la tâche à laquelle le processeur a été alloué afin qu'elle s'exécute. A un instant donné, une seule tâche peut occuper cet état. Au contraire, les deux états suivants peuvent être adoptés simultanément par plusieurs tâches.
- **Prêt** : La tâche demande à être exécutée et attend que l'ordonnanceur l'élise en la plaçant à l'état *courant*.
- **Suspendu** : La tâche est passive et passera à l'état *prêt* à sa prochaine activation.

Le standard OSEK prévoit, pour les **tâches** dites **étendues**, un état *"en attente"* correspondant à l'attente d'un événement comme la réception d'un message. Toutefois, cette thèse ne s'intéresse qu'aux **tâches** dites **basiques** et se limite donc aux trois états énumérés ci-dessus.

Selon le standard OSEK, une tâche ne peut se terminer sans appeler l'un des services *TerminateTask* ou *ChainTask*. Le service *TerminateTask* termine proprement la tâche en la replaçant dans l'état *suspendu* et élit la tâche qui lui succédera à l'état *courant*. Le service *ChainTask* équivaut au service *TerminateTask* mais permet à la tâche qui se termine d'activer une autre tâche ou même de se réactiver ; la tâche ainsi activée sera prise en compte par l'ordonnanceur lors de l'élection de la nouvelle tâche courante. Une tâche peut également activer une autre tâche, ou se réactiver elle-même, à l'aide du service *ActivateTask*. Cependant, tout au long de cette thèse, les tâches périodiques indépendantes ne sont jamais activées à l'aide des services *ChainTask* et *ActivateTask* mais uniquement par des alarmes dont le mécanisme est détaillé à la section 3.d. De plus, ces tâches périodiques indépendantes se termineront toujours en appelant le service *TerminateTask*. La figure 3.2 et le tableau 3.1 illustrent les transitions entre les états énumérés ci-dessus.

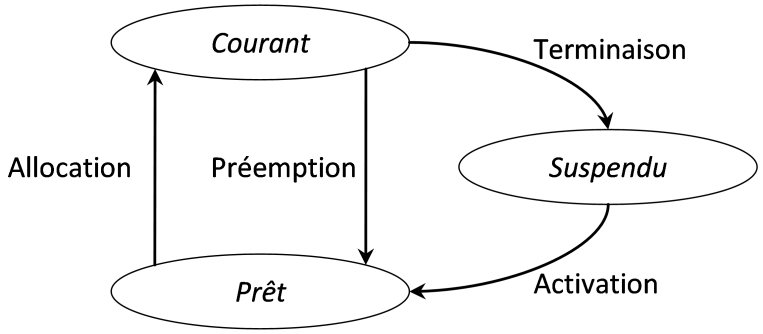

FIGURE 3.2 – Illustration des transitions entre les différents états d'une tâche basique

Transition	*Ancien état*	*Nouvel état*	*Description*
Activation	Suspendu	Prêt	Une nouvelle tâche est placée à l'état *prêt* suite à son activation par l'un des services *ActivateTask* ou *ChainTask*, ou bien par une alarme.
Allocation	Prêt	Courant	Une tâche à l'état *prêt* se voit attribuer l'utilisation du processeur par l'ordonnanceur et passe à l'état *courant*.
Préemption	Courant	Prêt	La tâche à l'état *courant* perd l'utilisation du processeur au profit d'une autre tâche, plus prioritaire, élue par l'ordonnanceur.
Terminaison	Courant	Suspendu	La tâche à l'état *courant* se termine en appelant l'un des services *TerminateTask* ou *ChainTask*.

TABLE 3.1 – Descriptions des transitions entre les différents états d'une tâche basique

3.c Problème du partage de ressource

Supposons que plusieurs tâches doivent accéder à une même ressource qui ne peut être partagée comme une mémoire ou un périphérique de communication. Afin d'ordonner l'exploitation de ce type de ressources, des mécanismes de gestion des accès concurrents ont été définis. La sous section 3.c.i illustre un mécanisme basé sur des sémaphores ainsi que le risque d'interblocage inhérent à cette technique. La méthode du plafond de priorité, imposée par le standard OSEK, est exposée à la sous section 3.c.ii. Nous y montrons comment cette méthode évite tout risque d'interblocage et s'avère ainsi plus fiable que l'emploi de sémaphores.

3.c.i Cas d'interblocage

Un sémaphore binaire, encore appelé sémaphore d'exclusion mutuelle ou mutex, est associé à une ressource afin d'en limiter l'accès à une seule tâche à un instant donné. Initialisé avec un compteur à 1, le sémaphore binaire requiert deux primitives :

– **Prendre** : Lorsqu'une tâche utilise une ressource protégée par un sémaphore, toute section de son code liée à cette ressource commence par *prendre le sémaphore* qui la protège. Cette primitive vérifie si le compteur du sémaphore est à 1. Si tel est le cas, la ressource est libre et la tâche courante peut y accéder. Au niveau du sémaphore, cet accès à la ressource est matérialisé par la décrémentation de son compteur. Autrement, la nullité du compteur de ce sémaphore signifie que la ressource est déjà exploitée par une autre tâche ; la tâche courante ne peut alors pas y accéder. Dans ce cas, le sémaphore place la tâche courante dans une file d'attente, qui lui est associée, en attendant que la ressource soit libérée.

– **Vendre** : Une fois l'utilisation de la ressource autorisée par le sémaphore qui la protège, la tâche qui en profite devra *vendre ce sémaphore* à la fin de la section de son code liée à cette ressource. La primitive *vendre* du sémaphore commence par vérifier si sa file d'attente est vide ou non. Si la file d'attente du sémaphore est vide, cela signifie qu'aucune autre tâche n'a demandé l'accès à la ressource ; il n'y a alors aucune tâche en attente de celle-ci. Dans ce cas, la primitive *vendre* se contente de remettre le compteur du sémaphore à 1 pour permettre un nouvel accès à la ressource. Au contraire, si la file d'attente du sémaphore n'est pas vide, celle-ci représente l'ensemble des tâches ayant demandé un accès à la ressource ; la primitive *vendre* va alors accorder la ressource à la tâche qui attend depuis le plus longtemps parmi les tâches les plus prioritaires en attente. De plus, l'ordonnanceur peut éventuellement élire la tâche qui vient juste de récupérer la ressource.

Assurément, ce mécanisme limite l'accès à la ressource à une seule tâche à tout instant, et répond ainsi parfaitement à son premier objectif. Cependant, sa mise en oeuvre nécessite une grande rigueur de la part du concepteur afin d'éviter tout **interblocage** comme illustré à la figure 3.3.

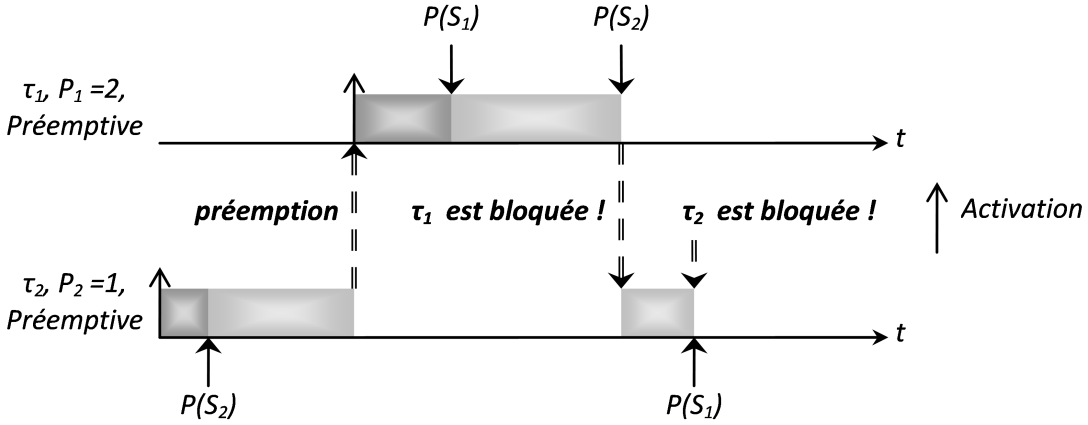

FIGURE 3.3 – Cas d'interblocage avec deux sémaphores binaires

La figure 3.3 montre un scénario où deux tâches τ_1 et τ_2 se partagent deux ressources protégées à l'aide de deux sémaphores binaires S_1 et S_2. La tâche τ_1 tente de *prendre* S_1 puis S_2, alors que la tâche τ_2 cherche à *prendre* S_2 puis S_1. Activée en première, la tâche τ_2 a le temps de *prendre* S_2 avant d'être préemptée par la tâche τ_1. Ayant obtenu le processeur, la tâche τ_1 *prend* S_1 puis tente de *prendre* S_2. Le sémaphore S_2 ayant déjà été pris par la tâche τ_2, la tâche τ_1 se retrouve bloquée et rend le processeur à la tâche τ_2. La tâche τ_2, reprennant son exécution, essaye de *prendre* le sémaphore S_1 déjà pris par la tâche τ_1 et se retrouve à son tour bloquée. Finalement, les deux tâches τ_1 et τ_2 se retrouvent toutes deux bloquées.

Pour assurer l'absence d'interblocage, le concepteur de l'application doit garantir qu'aucune tâche ne puisse utiliser une ressource sans s'exécuter à un niveau de priorité supérieur ou égal à celui de toute autre tâche accédant à cette ressource. Le mécanisme du plafond de priorité garantit cette condition et évite ainsi tout interblocage dans l'application.

3.c.ii Méthode du plafond de priorité

Afin d'éviter tout problème d'interblocage, la norme OSEK s'appuie sur la méthode du plafond de priorité : chaque ressource se voit attribuer une priorité appelée priorité de plafond. Cette priorité de plafond est supérieure ou égale à la priorité maximale observée sur l'ensemble des tâches accédant à la ressource concernée. De plus, cette priorité de plafond est strictement inférieure à la priorité minimale observée sur l'ensemble des tâches n'utilisant pas la ressource et de priorité strictement supérieure à toute tâche l'utilisant. Les deux primitives suivantes sont alors nécessaires aux tâches pour obtenir puis libérer une ressource :

- **GetResource** : Ce service accorde la ressource spécifiée en argument à la tâche courante. Afin d'éviter tout interblocage et d'empêcher que toute autre tâche n'accède à cette ressource, ce service augmente la priorité de la tâche courante en la plaçant à la priorité de plafond associée à la ressource en question. La priorité de plafond étant supérieure ou égale à la priorité maximale relevée sur l'ensemble des tâches utilisant la ressource, après modification de sa priorité, la tâche courante ne peut en aucun cas être préemptée par une tâche demandant l'accès à cette même ressource qui se trouve ainsi correctement protégée. Cependant, certaines tâches qui n'emploient pas cette ressource peuvent avoir une priorité strictement supérieure à sa priorité de plafond et préempter la tâche courante.
- **ReleaseResource** : Ce service libère la ressource spécifiée en argument en ramenant la priorité de la tâche courante à sa valeur antérieure. Une fois le changement de priorité effectué, une autre tâche peut éventuellement être élue. Cette élection et le changement de contexte associé sont alors effectués par ce même service.

La figure 3.4 illustre la méthode du plafond de priorité. Cette illustration comporte trois tâches préemptives τ_1, τ_2 et τ_3 ainsi que deux ressources R_1 et R_2. Nous précisons que la tâche τ_1 est plus prioritaire que la tâche τ_2, elle-même plus prioritaire que la tâche τ_3. De plus, la priorité de plafond associée à la ressource R_1 est supérieure à la priorité de la tâche τ_1. La priorité de plafond associée à la ressource R_2 est, quant à elle, inférieure à la priorité de la tâche τ_1 et supérieure à celle de la tâche τ_2.

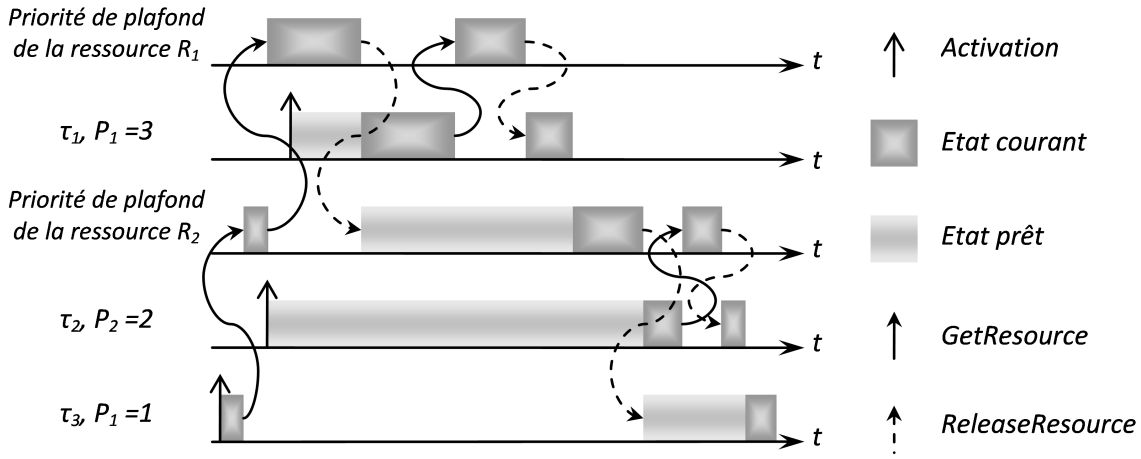

FIGURE 3.4 – Illustration de la méthode du plafond de priorité

Comme nous le voyons à la figure 3.4, la tâche τ_3 est la première activée et commence par récupérer la ressource R_2 à l'aide du service *GetResource*. Sa priorité devient alors supérieure à celle de la tâche τ_2 qui, une fois activée, ne peut s'exécuter. Puis la tâche τ_3 récupère la ressource R_1 et devient alors plus prioritaire que n'importe quelle autre tâche retardant ainsi l'exécution de la tâche τ_1. Dès que la tâche τ_3 libère la ressource R_1, en appelant le service *ReleaseResource*, sa priorité redescend à la priorité de plafond associée à la ressource R_2 et la tâche τ_1 entame son exécution pendant laquelle elle récupère puis libère la ressource R_1. Une fois l'exécution de la tâche τ_1 terminée, celle de la tâche τ_3 reprend. Lorsque la tâche τ_3 libère la ressource R_2 sa priorité revient à son niveau initial et la tâche τ_2 peut alors s'exécuter. Lors de son exécution, la tâche τ_2 récupère puis libère la ressource R_2. L'exécution de la tâche τ_3 redémarre quand celle de la tâche τ_2 s'achève. Finalement, l'exécution de la tâche τ_1 a été retardée par la tâche τ_3, pourtant initialement moins prioritaire, tant que celle-ci utilisait la ressource R_1. Quant à la tâche τ_2, son exécution a été également retardée par la tâche τ_3, là encore initialement moins prioritaire, pendant tout le temps où celle-ci disposait de la ressource R_2. Bien évidemment, l'exécution de la tâche τ_1 a contribué au retard pris par celle de la tâche τ_2 mais ceci se justifie pleinement par le fait que la tâche τ_1 est plus prioritaire que la tâche τ_2.

Notez que toute tâche τ_i peut être retardée par une autre tâche, initialement moins prioritaire, si celle-ci a eu le temps d'acquérir une ressource dont la priorité de plafond s'avère supérieure ou égale à la priorité de la tâche τ_i. Par ailleurs, la méthode du plafond de priorité garantit qu'une tâche ne puisse être retardée que par, au plus, une tâche moins prioritaire.

3.d Mécanisme des alarmes

Afin de déployer le modèle de tâche périodique, nous nous appuyons sur le mécanisme des alarmes proposé par la norme OSEK. Chacune de ces alarmes possède deux paramètres : la date de son premier déclenchement et sa période à partir de celui-ci. A chacun de ses déclenchements, une alarme peut activer une tâche, provoquer un événement, ou bien encore appeler une fonction de type *Alarmcallback*. Notez que toute fonction de type *Alarmcallback* agit de la même façon qu'une interruption de type ISR2 et interrompt la tâche courante qu'elle soit préemptive ou non. Aussi, l'action d'une alarme est identique lors de tous ses déclenchements et définie statiquement lors de la conception de l'application. La figure 3.5 illustre une alarme qui active périodiquement une tâche τ_j de période T_j. De plus, la première activation de la tâche τ_j survient à la date t_j.

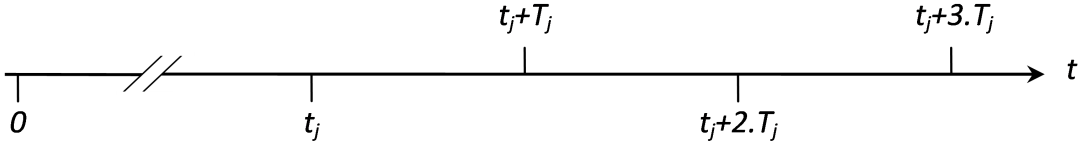

FIGURE 3.5 – Illustration des déclenchements d'une alarme

Bien évidemment, la notion de temps introduite par ce mécanisme implique que tout exécutif OSEK gère sa propre base de temps. Pour ce faire, une interruption périodique, de période T_{tick}, est mise en oeuvre à l'aide d'un compteur. Supposons maintenant que, lors de la conception d'une application, une période T_j égale à $11ms$ ait été attribuée à une tâche τ_j et que la période T_{tick} soit égale à $2ms$. Dans ce cas, T_j ne serait pas multiple de T_{tick}. Une fois l'application déployée, la période de la tâche τ_j serait alors égale à T_j^* et donnée par la formule suivante :

$$T_j^* = (1 + \lfloor \frac{\max(T_j - T_{tick}/2, 0)}{T_{tick}} \rfloor) T_{tick}$$

Ainsi, dans le cas où la période T_j ne serait pas multiple de T_{tick}, T_j serait arrondie à son plus proche multiple de T_{tick}. Dans l'hypothèse où T_j serait équidistante de deux multiples de T_{tick} consécutifs, T_j serait alors arrondie au multiple supérieur.

3.e Illustration et caractérisation des charges OSEK

Notre exécutif OSEK, fourni par la société Vector, est déployé sur un microcontrôleur dsPIC30F6014 fourni par la société Microchip. Après avoir observé, en simulation, sur des ensembles restreints de tâches, l'exécution de ce système d'exploitation, nous en avons identifié les charges, exposées à la sous section 3.e.i, nécessaires au bon ordonnancement des tâches ainsi qu'à la gestion des ressources. L'objectif visant à prévoir le temps de réponse pire cas de chaque tâche d'une application lors de sa conception, nous devons caractériser la durée d'exécution pire cas de chacune des charges précédentes. Une fois ces charges mesurées selon la méthode détaillée à la sous section 3.e.ii, une fonction de caractérisation peut être écrite pour chacune d'entre elles. Les résultats des mesures réalisées, ainsi que les fonctions de caractérisation, sont fournis à la sous section 3.e.iii.

3.e.i Illustration des charges OSEK

Tout d'abord, nous rappelons que tous les périphériques d'un microcontrôleur sont pilotés par sa fréquence interne, c'est à dire la fréquence à laquelle le microcontrôleur exécute son programme. Exprimée en "millions d'instructions par seconde", cette fréquence est naturellement la plus élevée au sein du microcontrôleur. En conséquence, le cycle interne, c'est à dire la période correspondant à la fréquence interne, apparaît comme l'unité de temps la plus précise à notre disposition. C'est pourquoi, dans la suite de cet exposé, nous utiliserons fréquemment le cycle interne comme unité temporelle et parlerons alors de *cycles*.

Comme expliqué à la section 3.d, le standard OSEK met en oeuvre des alarmes devant chacune se réveiller périodiquement à partir d'une date fixée par le concepteur du système. La gestion de ces alarmes nécessite une base de temps obtenue à l'aide d'une interruption périodique, de période T_{tick}, basée sur un compteur du microcontrôleur. Chaque occurence de cette interruption est traitée par le système d'exploitation qui, à cette occasion, actualise sa propre base de temps et gère les alarmes.

Comme défini à la sous section 2.a.v, le facteur d'utilisation du processeur pour la fonction associée à cette interruption est égal à $U_{tick} = C_{tick}/T_{tick}$, où C_{tick} représente la durée d'exécution pire cas de cette fonction. La figure 3.6 montre le temps de réponse d'une tâche, mesuré sur une plateforme réelle, en fonction du paramètre T_{tick}. Cette tâche ne comporte qu'une boucle vide telle que sa durée d'exécution pire cas soit fixée à $50ms$.

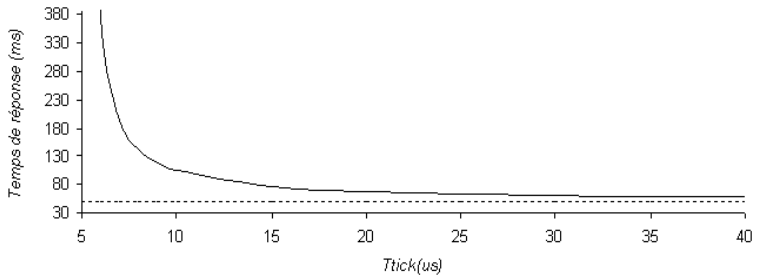

FIGURE 3.6 – Comparaison entre le WCET (en pointillés) et le temps de réponse d'une tâche, en fonction de T_{tick}

Nous constatons que le temps de réponse de la tâche augmente à mesure que le paramètre T_{tick} diminue. Ce temps de réponse dépasse le double de la durée d'exécution pire cas de la tâche à $T_{tick} = 10\mu s$ pour atteindre $3571ms$ à $T_{tick} = 5\mu s$. La diminution du paramètre T_{tick} augmente donc bien la charge du processeur et affecte notablement le temps de réponse de la tâche. Dans le cas d'une application réelle, un mauvais choix de ce paramètre nuirait alors à l'ensemble des tâches présentes.

Sachant que le système gère les alarmes dans la fonction d'interruption précédente, chaque alarme déclenchée exécutera son action immédiatement après celle-ci. Comme expliqué à la section 3.d, une alarme peut activer une tâche, provoquer un événement, ou bien encore appeler une fonction de type *Alarmcallback*. Dans notre étude, les alarmes sont employées exclusivement pour activer les tâches périodiques. Ainsi, après la fonction d'interruption, chaque alarme déclenchée va activer la tâche qui lui est associée. Nous noterons C_{act} la durée d'exécution pire cas de cette fonction d'activation. Supposons qu'au moins une tâche ait été activée suite à la fonction d'interruption et que le processeur soit attribué à l'une d'entre elles immédiatement après son activation. Dans ce cas, nous noterons C_{sched} la durée d'exécution pire cas de la fonction en charge du changement de contexte qui intervient lorsqu'une tâche est élue immédiatement après son activation. Reste que toute tâche doit un jour se terminer. Notre étude ne portant que sur des tâches périodiques indépendantes, chaque tâche se termine en appelant le service *TerminateTask* dont la durée d'exécution pire cas est notée C_{term}. Comme nous l'expliquons à la section 3.b, à chaque appel du service *TerminateTask*, celui-ci prend en charge l'élection de la prochaine tâche courante ainsi que le changement de contexte associé. Ainsi, le service *TerminateTask* est modélisé uniquement par une durée C_{term} et ne requière aucune durée C_{sched} supplémentaire. Enfin, dans le cas où une ressource serait protégée avec la méthode du plafond de priorité, les services *GetResource* et *ReleaseResource* sont caractérisés par des durées d'exécution pires cas notées respectivement C_{get} et C_{rel}. Nous précisons à la sous section 3.c.ii que le service *ReleaseResource* est chargé du changement de contexte dans le cas où une nouvelle

tâche serait élue après que la priorité de celle à l'état *courant* ait été ramenée à sa valeur antérieure. Ainsi, comme le service *TerminateTask*, le service *ReleaseResource* n'engendre aucune durée C_{sched} et chacune de ses exécutions revient à une unique durée C_{rel}. Le tableau 3.2 récapitule l'ensemble des charges énoncées dans cette section.

Symbole	*Description*
C_{tick}	Durée d'exécution pire cas de la fonction d'interruption périodique pendant laquelle le système d'exploitation actualise sa propre base de temps et gère les alarmes.
C_{act}	Durée d'exécution pire cas de la fonction d'activation d'une tâche. Cette fonction est appelée lorsqu'une alarme retentit et active la tâche qui lui est associée. Si la tâche concernée était à l'état *suspendu*, elle passe à l'état *prêt*.
C_{sched}	Durée d'exécution pire cas de la fonction de changement de contexte qui intervient lorsqu'une tâche se voit attribuer le processeur immédiatement après avoir été activée par son alarme. Cette tâche se retrouve alors à l'état *courant*.
C_{term}	Durée d'exécution pire cas du service *TerminateTask* appelé à la fin de chaque tâche. Ce service est en charge d'élire, parmi l'ensemble des tâches à l'état *prêt*, si tant est que celui-ci ne soit pas vide, la prochaine tâche courante et d'effectuer le changement de contexte nécessaire à son exécution. Cette élection ainsi que le changement de contexte étant alors entièrement à la charge du service *TerminateTask*, celui-ci ne nécessite aucune durée C_{sched} supplémentaire.
C_{get}	Durée d'exécution pire cas du service *GetResource*. Ce service permet à une tâche qui en a besoin d'accéder à une ressource et modifie la priorité de cette tâche en la remplaçant par la priorité de plafond associée à la ressource.
C_{rel}	Durée d'exécution pire cas du service *ReleaseResource*. Lorsqu'une tâche utilise une ressource, après l'avoir obtenue grâce au service *GetResource*, celle-ci doit impérativement la libérer en appelant le service *ReleaseResource* qui ramène sa priorité à sa valeur antérieure. Une fois sa priorité modifiée, la tâche peut perdre le processeur. Le service *ReleaseResource* est alors en charge d'élire la nouvelle tâche courante et de procéder au changement de contexte. Ainsi, ce service n'engendre aucune durée C_{sched} additionnelle.

TABLE 3.2 – Description des différentes charges dues au système d'exploitation

La figure 3.7 illustre les interférences créées par les charges précédentes à l'aide d'une application à quatre tâches parmi lesquelles seules τ_2 et τ_3 sont préemptives. De plus, la tâche τ_1 est moins prioritaire que la tâche τ_2, elle même moins prioritaire que la tâche τ_3, elle même moins prioritaire que la tâche τ_4. La tâche τ_4 n'utilisant pas la ressource, la priorité de plafond associée à celle-ci est inférieure à la priorité de cette tâche. Ainsi, lorsque les tâches τ_2 et τ_3 utiliseront la ressource, celles-ci pourront tout de même être préemptées par la tâche τ_4.

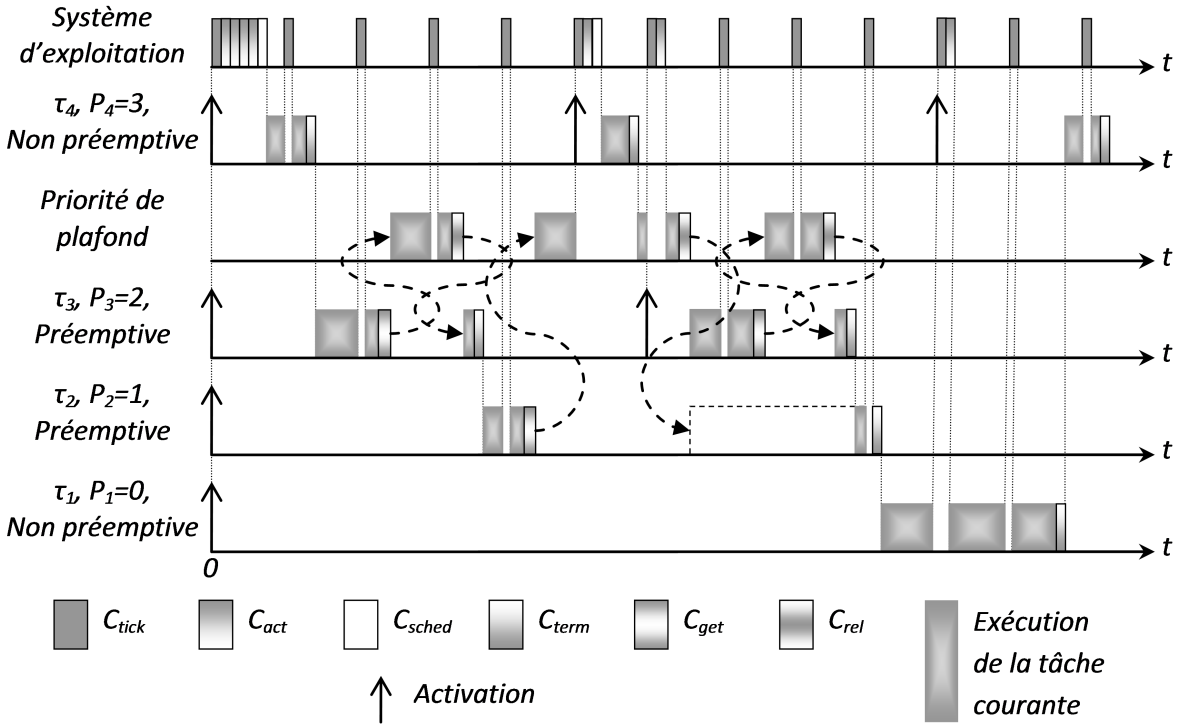

FIGURE 3.7 – Exécution réelle d'une application à quatre tâches utilisant une ressource

La figure 3.7 montre, au niveau du système d'exploitation, que la charge C_{tick} apparaît périodiquement et interrompt systématiquement la tâche à l'état *courant* que celle-ci soit préemptive ou non. En effet, comme nous l'avons dit à la section 3.a, le système d'exploitation est plus prioritaire que n'importe quelle tâche. Nous rappelons que durant la charge C_{tick}, le système d'exploitation maintient sa base de temps et gère l'ensemble des alarmes. C'est pourquoi, les charges C_{act}, symbolisant chacune l'activation d'une tâche suite au retentissement de l'alarme associée, font toujours suite à une charge C_{tick}.

Suite à la première charge C_{tick}, nous observons quatre charges C_{act} correspondant aux activations simultanées des quatre tâches. Une fois les quatre tâches activées, la tâche τ_4 étant la plus prioritaire, celle-ci se voit naturellement attribuer le processeur. La charge C_{sched} qui suit les quatre charges C_{act} correspond alors au changement de contexte nécessaire à l'exécution de la tâche τ_4 et marque son passage à l'état *courant*. Nous remarquons que la première exécution de la tâche τ_4 est interrompue par une charge C_{tick} et se termine par une charge C_{term} correspondant à son appel au service *TerminateTask*. Nous rappelons

que ce service est en charge de terminer proprement la tâche qui l'appelle mais, également, d'élire, parmi les tâches à l'état *prêt*, la prochaine qui se verra allouer le processeur ainsi que d'effectuer le changement de contexte nécessaire à son exécution. Le coût de ce changement de contexte étant comptabilisé au niveau de la charge C_{term} aucune charge C_{sched} supplémentaire n'est nécessaire. Lorsque la tâche τ_4 appelle le service *TerminateTask*, la tâche τ_3 étant plus prioritaire que les tâches τ_1 et τ_2, celle-ci se voit attribuer le processeur.

Au début de son exécution, la tâche τ_3 est préemptée une première fois par le système d'exploitation afin qu'une charge C_{tick} s'exécute. Puis, la tâche τ_3 appelle le service *GetResource* pour utiliser la ressource. Cet appel correspond à la charge C_{get} après laquelle la tâche τ_3 poursuit son exécution avec une priorité égale à la priorité de plafond de la ressource. Pendant son utilisation de la ressource, la tâche τ_3 est à nouveau préemptée par une charge C_{tick} du système d'exploitation. Dès que la tâche τ_3 libère la ressource grâce au service *ReleaseResource*, représenté par la charge C_{rel}, sa priorité retrouve sa valeur initiale. Enfin, la tâche τ_3 se termine avec une charge C_{term}. La tâche τ_2 est ensuite élue et obtient le processeur.

Après avoir été préemptée par une première charge C_{tick} du système d'exploitation, la tâche τ_2 appelle le service *GetResource*. Bien que sa priorité soit alors égale à la priorité de plafond associée à la ressource, la tâche τ_2 est à nouveau préemptée par le système d'exploitation qui exécute alors une charge C_{tick}, une charge C_{act} correspondant à l'activation de la tâche τ_4 et une charge C_{sched} chargée du changement de contexte nécessaire à l'exécution de la tâche τ_4. En effet, la priorité de la tâche τ_4 étant supérieure à la priorité de plafond de la ressource, celle-ci peut donc préempter la tâche τ_2 alors que celle-ci utilise cette même ressource. La tâche τ_2 récupère le processeur après que la tâche τ_4 se soit terminée avec une charge C_{term}. Alors que la tâche τ_2 poursuit son exécution et utilise la ressource, celle-ci est une nouvelle fois préemptée par le système d'exploitation qui exécute une charge C_{tick} puis une charge C_{act} correspondant alors à l'activation de la tâche τ_3. La tâche τ_3 ne pouvant pas préempter la tâche τ_2, aucun changement de contexte ni, en conséquence, aucune charge C_{sched} ne sont nécessaires. La tâche τ_2 reprend ensuite son exécution et libère la ressource à l'aide du service *ReleaseResource*. Une fois sa priorité revenue à sa valeur initiale, la tâche τ_2 est préemptée par la tâche τ_3 qui obtient l'utilisation du processeur.

Lors de sa seconde exécution, la tâche τ_3 est préemptée deux fois par le système d'exploitation, à chaque fois pour exécuter une charge C_{tick}. Durant cette exécution, la tâche τ_3 va acquérir puis libérer la ressource et enfin se terminer grâce au service *TerminateTask* qui va alors redonner le processeur à la tâche τ_2.

Pendant la fin de son exécution, la tâche τ_2 est préemptée une fois par une charge C_{tick} puis se termine en appelant le service *TerminateTask* qui va alors attribuer le processeur à la tâche τ_1.

Bien que non préemptive, la tâche τ_1 est préemptée deux fois par le système d'exploitation durant son exécution. La première fois, le système d'exploitation exécute une charge C_{tick} puis une charge C_{act} signifiant l'activation de la tâche τ_4. La tâche τ_1 étant non préemptive, celle-ci ne peut être préemptée par la tâche τ_4 pourtant plus prioritaire. C'est pourquoi, aucune charge C_{sched} n'apparaît après l'activation de la tâche τ_4. La seconde fois, le système d'exploitation se contente d'exécuter une charge C_{tick}. Après cela, la tâche τ_1 poursuit son exécution et se termine avec une charge C_{term}. Le processeur est ensuite attribué à la tâche τ_4. Celle-ci verra alors son exécution préemptée une fois par une charge C_{tick} du système d'exploitation et terminée par une charge C_{term}. Cette dernière exécution de la tâche τ_4 est donc retardée par une tâche non préemptive et moins prioritaire ce qui illustre l'effet non préemptif présenté au paragraphe 2.a.iii.2.

3.e.ii Mesure des charges OSEK

Maintenant que les charges dues au système d'exploitation sont connues, l'objectif consiste à mesurer la durée d'exécution pire cas de chacune d'entre elles afin de pouvoir les intégrer aux conditions de faisabilité classiques. La méthode utilisée pour mesurer ces durées est détaillée au paragraphe 3.e.ii.1. Le protocole, décrivant les ensembles de tâches mis en oeuvre pour ces mesures, est ensuite détaillé au paragraphe 3.e.ii.2.

3.e.ii.1 Méthode pour la mesure des durées pire cas d'exécution

Afin de pouvoir mesurer les durées d'exécution de n'importe quelle fonction ou tâche, deux services nommés *SpyStart* et *SpyStop* ont été implémentés. Avant l'exécution de l'application, nous attribuons un identifiant à chaque fonction ou tâche dont les durées d'exécution doivent être mesurées. Passé en argument, cet identifiant permet aux services précédents de classer correctement les mesures effectuées. Nous définissons également le nombre N de mesures à effectuer pour chaque durée.

Pour accomplir les mesures désirées, les services *SpyStart* et *SpyStop* nécessitent un autre compteur différent de celui utilisé par le système d'exploitation pour créer sa base de temps. Afin d'obtenir les mesures les plus justes possibles, nous avons implémenté ces deux services tels que :

- N mesures sont effectuées pour chaque identifiant, c'est à dire pour chaque fonction ou tâche que nous voulons étudier.
- Le service *SpyStart* déclenche le chronomètre juste avant de se terminer de sorte à ce que son exécution fausse le moins possible la mesure.
- Le service *SpyStop* commence par arrêter le chronomètre de sorte à ce que son exécution fausse le moins possible la mesure.
- Le service *SpyStop* envoie l'ensemble des mesures effectuées sur le port série dès la dernière mesure du dernier identifiant terminée.

Afin de pouvoir mesurer les temps de réponse de chaque tâche, ces deux services ont ensuite été intégrés au système d'exploitation. Aussi, leur temps d'exécution est compris dans les mesures des charges dues au noyau OSEK présentées à la sous section 3.e.iii.

3.e.ii.2 Protocole pour la mesure des charges dues au système d'exploitation

Avant d'exposer le protocole que nous avons suivi pour mesurer les charges de notre exécutif OSEK, nous expliquons maintenant comment celui-ci est mis en oeuvre dans la pratique. Lors de la conception d'une application, nous utilisons un logiciel appelé "OIL Configurator" qui nous permet d'en décrire les différents objets tels que les ressources, les alarmes ou encore les tâches. Le tableau 3.3 récapitule les différents paramètres que nous devons spécifier au "OIL Configurator" pour chaque objet de l'application en fonction de son type :

Type d'objet	*Paramètre(s)*
Ressource	Nom de la ressource
Alarme	Période
	Date du premier déclenchement
Tâche	Priorité
	Tâche préemptive ou non préemptive

TABLE 3.3 – Paramètres des différents objets d'une application OSEK

Bien évidemment, chaque application peut posséder un nombre différent d'objets de chaque type. Pour des raisons de temps, nous n'avons mesuré les charges C_{get} et C_{rel} qu'avec des ensembles de tâches utilisant une seule ressource. Chaque tâche étant associée à une alarme chargée de l'activer périodiquement, nous aurons toujours autant d'alarmes que de tâches. De plus, les six charges du système d'exploitation sont mesurées avec des ensembles de n tâches où n appartient à l'intervalle $[1, 20]$. Finalement, les mesures réalisées seront exploitables avec des ensembles comportant une à vingt tâches et utilisant au plus une ressource.

Une fois l'application intégralement décrite, le "OIL Configurator" génère le système d'exploitation correspondant. Aussi, le code du système généré peut différer d'une application à une autre. A l'évidence, le nom donné à la ressource ne bouleversera pas le code généré. Les paramètres des alarmes, quant à eux, ne jouent que sur les dates auxquelles celles-ci surviennent. Par contre, les paramètres des tâches peuvent avoir un impact sur le code généré. C'est pour cette raison que, dans le protocole suivant, nous étudions, entre autres, l'ordonnancement FIFO dans les contextes préemptif, non préemptif et mixte. En effet, comme nous le soulignons à la sous section 2.d.ii, avec un ordonnancement FIFO, toutes les tâches ayant la même priorité fixe, la nature préemptive ou non d'une tâche ne modifie en rien son ordonnancement. Cependant, nous n'avons aucune certitude quant à l'impact de ce paramètre sur le code généré et, par conséquent, sur les charges de l'exécutif.

L'ensemble du protocole est décrit par les deux paragraphes suivants. Le premier décrit des ensembles de tâches, toujours activées simultanément, utilisant toutes une ressource commune et mis en oeuvre pour mesurer les six charges détaillées à la sous section 3.e.i. Lorsque toutes les tâches sont activées simultanément, celles-ci sont exécutées chacune leur tour selon leur priorité et, dans le cas où plusieurs posséderaient la même priorité, leur date d'activation. C'est pourquoi, le second paragraphe expose de nouveaux ensembles de

tâches, utilisant là encore une même ressource, mais dont les activations sont échelonnées de sorte à ce que les tâches les plus prioritaires soient activées pendant que la tâche la moins prioritaire utilise la ressource. Ainsi, le mécanisme du plafond de priorité va retarder l'exécution des tâches les plus prioritaires. Dans ce cas, lors de la libération de la ressource, le service *ReleaseResource* élit la prochaine tâche devant utiliser le processeur. Ainsi, ces nouveaux ensembles de tâches mettent en avant le mécanisme du plafond de priorité exposé à la sous section 3.c.ii et permettent de peaufiner la mesure des charges C_{get} et C_{rel}.

Ensembles de tâches avec activations simultanées

Nous commençons donc par présenter six ensembles de n tâches, avec $n \in [1, 20]$, dont les activations arriveront toujours simultanément. En effet, dans chacun de ces ensembles, toutes les tâches possèdent la même période T et sont activées simultanément à partir de la date 0. De plus, toutes les tâches se contentent de prendre puis de libérer la ressource à l'aide des services *GetResource* et *ReleaseResource*. La figure 3.8 illustre un ordonnancement FIFO de trois tâches conformes aux spécifications précédentes.

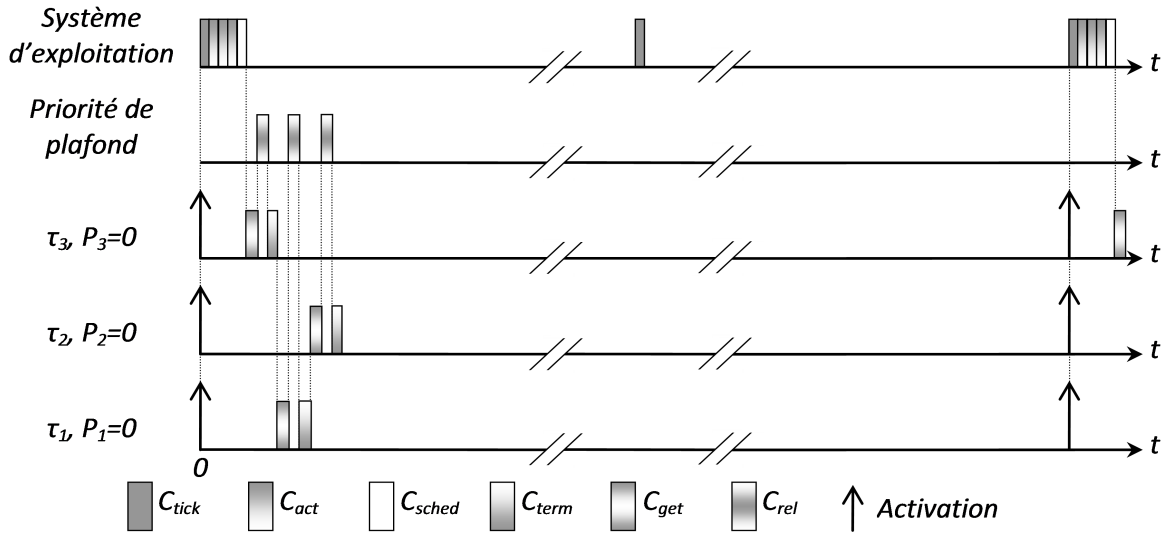

FIGURE 3.8 – Ordonnancement FIFO de trois tâches activées simultanément

Nous observons, à la figure 3.8, qu'au niveau du système d'exploitation, la première charge C_{tick} est suivie de trois charges C_{act} correspondant aux activations simultanées des trois tâches. Les trois tâches ayant la même priorité, la tâche τ_3 est arbitrairement élue et la charge C_{sched} effectue le changement de contexte nécessaire à son exécution. Son exécution entamée, la tâche τ_3 récupère immédiatement la ressource puis la libère de suite. Enfin, la tâche τ_3 se termine avec une charge C_{term} durant laquelle le processeur est arbitrairement alloué à la tâche τ_1 et le changement de contexte effectué. Toutes les tâches ayant le même comportement, à son tour, la tâche τ_1 récupère la ressource, la libère et se termine avec une charge C_{term}. Enfin, la tâche τ_2 se voit attribuer le processeur et s'exécute selon le même motif. Nous soulignons que cette illustration reste valable que le contexte soit préemptif, non préemptif ou mixte. Par ailleurs, dans le cas d'un ordonnancement FP, la seule

différence tient au fait que les tâches ne seraient plus exécutées dans un ordre arbitraire mais suivant leur priorité. Enfin, nous remarquons qu'une charge C_{tick} apparaît entre les deux instants critiques. Ceci tient au fait que nous avons fixé la valeur T_{tick} à $T/2$. Ainsi, nous obtenons les deux cas extrêmes : une charge C_{tick} durant laquelle toutes les alarmes sont déclenchées et une autre où aucune ne l'est. Bien évidemment, le premier cas est le plus important car c'est celui qui produit la durée d'exécution maximale de la charge C_{tick}.

Chaque ensemble de n tâches aboutit donc bien à la durée d'exécution maximale de la charge C_{tick} en déclenchant simultanément toutes les alarmes. La durée de l'élection de la première tâche courante est également maximisée puisqu'exécutée parmi n tâches. Cependant, nous ignorons, a priori, si cette élection incombe à la charge C_{act} ou à la charge C_{sched}. Ayant un nombre maximal de tâches activées simultanément, nous serons également dans le pire cas vis-à-vis de la charge C_{term} qui, lors de sa première apparition après chaque instant critique, devra élire la prochaine tâche courante parmi $n-1$ tâches à l'état *prêt*. Finalement, ces ensembles apparaissent bien adaptés pour la mesure des charges C_{tick}, C_{act}, C_{sched} et C_{term}. Les charges C_{get} et C_{rel} sont également mesurées mais nous compléterons leurs mesures au sous paragraphe suivant.

Une fois activée, chaque tâche se contente d'acquérir la ressource, de la libérer puis de se terminer. Ainsi, nous pouvons dire que chaque tâche engendre une charge de travail égale à $C_{act}+C_{get}+C_{rel}+C_{term}$. Sachant que, sur chaque période T, nous avons également deux charges C_{tick} et une charge C_{sched}, la charge totale de travail pour le processeur, lorsque $n = 20$, sur une période T, vaut $C_{sched}+2\times C_{tick}+20\times(C_{act}+C_{get}+C_{rel}+C_{term})$. Aussi, nous avons fixé la période T à 100000 *cycles* afin d'être assurés que toutes les charges aient toujours fini de s'exécuter lors du prochain instant critique. Autrement dit, la période T vérifie :

$$C_{sched} + 2 \times C_{tick} + 20 \times (C_{act} + C_{get} + C_{rel} + C_{term}) < T$$

Le tableau 3.4 récapitule les paramètres de toute tâche τ_i appartenant à l'un des six ensembles de n tâches devant être testés. Ces six ensembles de n tâches correspondent donc aux ordonnancements FIFO et FP appliqués à chacun des trois contextes : préemptif, non préemptif et mixte. Nous rappelons que chacun de ces six ensembles doit être mis en oeuvre vingt fois pour valider n de 1 à 20.

Notez que, dans ce paragraphe, nous n'examinons pas de cas FP/FIFO car approfondir l'ensemble des combinaisons possibles nous demanderait beaucoup trop de temps. Nous nous en tenons donc aux deux cas extrêmes et choisirons, pour chaque charge, le pire de ces deux cas lors de l'utilisation de nos conditions de faisabilité avec toute application ordonnancée FP/FIFO.

Ensemble de tâche pour l'ordonnancement :	*Priorité*	*Ordonnancement*
FIFO en contexte préemptif	$P_i = 0$	τ_i est préemptive
FIFO en contexte non préemptif	$P_i = 0$	τ_i est non préemptive
FIFO en contexte mixte	$P_i = 0$	τ_i est préemptive si i est pair et non préemptive autrement
FP en contexte préemptif	$P_i = i - 1$	τ_i est préemptive
FP en contexte non préemptif	$P_i = i - 1$	τ_i est non préemptive
FP en contexte mixte	$P_i = i - 1$	τ_i est préemptive si i est pair et non préemptive autrement

TABLE 3.4 – Paramètres de la tâche $\tau_i \in \{\tau_1, ..., \tau_n\}$ pour chacun des six ensembles de n tâches activées simultanément avec $n \in [1, 20]$

Ensembles de tâches avec activations échelonnées

Nous présentons maintenant trois nouveaux ensembles de n tâches avec $n \in [1, 20]$. Comme au sous paragraphe précédent, toutes les tâches se contentent de prendre puis de libérer la ressource à l'aide des services *GetResource* et *ReleaseResource*. La différence avec les six ensembles décrits précédemment tient au fait que les tâches ne seront plus activées simultanément. En effet, lorsqu'elles sont activées simultanément, les tâches s'exécutent chacune leur tour en fonction de leur priorité ou de leur date d'activation selon que l'ordonnancement soit FP ou FIFO. Dans ce cas, nous n'avons jamais d'accès concurrentiels à la ressource. Ainsi, le mécanisme du plafond de priorité n'est pas employé dans ses pires conditions. Nous considérons maintenant que la tâche τ_1, qui possède la plus faible priorité, est toujours activée en première et prend la ressource. Ainsi, grâce au mécanisme du plafond de priorité, l'exécution des tâches τ_2 à τ_n est retardée par la tâche τ_1 pourtant initialement moins prioritaire. Puis, lors de la libération de la ressource, si la tâche τ_1 est préemptive, le service *ReleaseResource* va élire, parmi les $n-1$ tâches plus prioritaires, la nouvelle tâche courante et effectuer le changement de contexte nécessaire à son exécution. Dans le cas où la tâche τ_1 ne serait pas préemptive, les autres tâches devraient normalement attendre la fin de son exécution et son appel au service *TerminateTask*.

Ainsi, ces nouveaux ensembles de tâches mettent en avant le mécanisme du plafond de priorité exposé à la sous section 3.c.ii et enrichissent les mesures déjà effectuées au paragraphe précédent, en particulier pour la charge C_{rel} qui, pour la première fois, se retrouve en situation d'élire la prochaine tâche active parmi un nombre maximum de tâches.

Pour mettre en oeuvre ces ensembles de tâches, et nous assurer que les activations des tâches τ_2 à τ_n auront bien lieu pendant que la tâche τ_1 utilise la ressource, deux solutions s'offrent à nous. La première consiste à associer, à chaque tâche $\tau_i \in \{\tau_1, ..., \tau_n\}$, une alarme de période T et dont la date du premier déclenchement, notée a_i, est fixée à $a_i = (i-1) \times T_{tick}$. Dans le cas où $n = 20$, la tâche τ_1, activée à la date 0, devra encore être en train d'utiliser la ressource lorsque la tâche τ_{20} sera activée à la date $19 \times T_{tick}$. Ainsi, la tâche τ_1 ne peut se contenter de prendre puis de libérer la ressource. Entre ses

appels aux services *GetResource* et *ReleaseResource*, une boucle vide d'une durée égale à $n \times T_{tick}$ est nécessaire. Cette solution accroît donc fortement la durée d'exécution pire cas de la tâche τ_1 ce qui s'avère pénalisant pendant les mesures. Qui plus est, bien qu'il soit possible de diminuer la valeur de T_{tick}, celle-ci doit rester supérieure à la durée nécessaire à l'exécution des autres charges. C'est pourquoi, nous avons opté pour la seconde solution qui consiste à activer périodiquement la tâche τ_1 à l'aide d'une alarme, de période T, dont la date du premier déclenchement est 0. Une fois activée, la tâche τ_1 récupère la ressource, active elle-même les autres tâches en utilisant exceptionnellement le service *ActivateTask*, libère la ressource et se termine. Les autres tâches se contentent, quant à elles, de prendre puis de libérer la ressource grâce aux services *GetResource* et *ReleaseResource*.

Une fois activées les tâches τ_2 à τ_n doivent attendre que la tâche τ_1 ait libéré la ressource pour entamer leur exécution. Ainsi, cette fois encore, la seule différence entre les ordonnancements FP et FIFO tient à l'ordre dans lequel ces tâches vont s'exécuter. Ces deux politiques aboutissant au même résultat vis-à-vis des charges C_{get} et C_{rel}, nous avons choisi arbitrairement de ne mettre en oeuvre que l'ordonnancement FP. Ainsi, les trois nouveaux ensembles, présentés ci-dessous, correspondent à la politique FP en contextes préemptif, non préemptif et mixte. La figure 3.9 illustre le scénario décrit précédemment avec trois tâches préemptives.

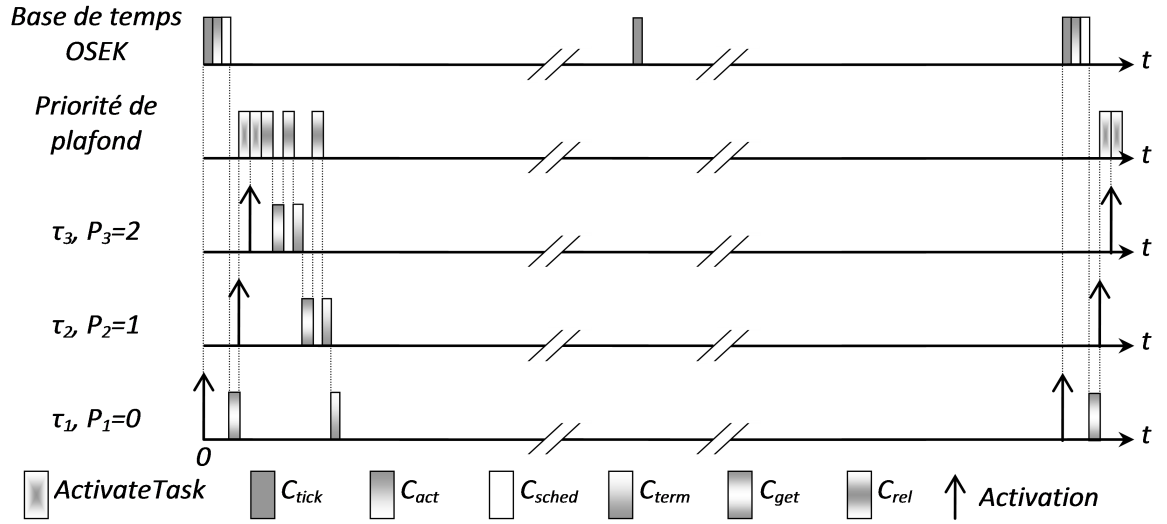

FIGURE 3.9 – Ordonnancement FP de trois tâches préemptives avec activations échelonnées

Nous observons, à la figure 3.9, qu'au niveau du système d'exploitation, la première charge C_{tick} est suivie d'une charge C_{act} correspondant à l'activation de la tâche τ_1 par son alarme. Le processeur étant libre à cet instant, la tâche τ_1, seule tâche à l'état *prêt*, est naturellement élue. La charge C_{sched}, qui apparaît ensuite, procède au changement de contexte nécessaire à l'exécution de la tâche τ_1. Ayant obtenu le processeur, l'exécution de la tâche τ_1 commence par une charge C_{get}. Une fois la ressource acquise, la tâche τ_1 poursuit son exécution avec une priorité égale à la priorité de plafond associée à cette ressource. Deux appels au service *ActivateTask*, nécessaires pour activer les tâches τ_2 et τ_3, apparaissent alors. Puis, la charge C_{rel} indique que la tâche τ_1 libère la ressource ce qui va permettre

aux tâches τ_2 et τ_3 d'entamer leur exécution. Durant cette charge C_{rel}, la tâche τ_3 est élue et le changement de contexte est effectué. Notez que la tâche τ_1 n'a pas encore terminé son exécution, celle-ci est donc bien préemptée par la tâche τ_3. Durant son exécution, la tâche τ_3 prend la ressource, la libère et se termine. La charge C_{term} qui clôture son exécution attribue ensuite le processeur à la tâche τ_2 qui s'exécute en suivant le même motif. Enfin, la charge C_{term} exécutée par la tâche τ_2 rend la main à la tâche τ_1 qui peut alors terminer son exécution. Nous précisons que cette illustration demeure valable pour les contextes non préemptif et mixte. Seule différence, dans le cas où la tâche τ_1 serait non préemptive, les autres tâches ne s'exécuteraient alors pas après que la tâche τ_1 ait libéré la ressource mais une fois celle-ci terminée.

Sur une période T, au niveau du système d'exploitation, nous observons une charge de travail égale à $2 \times C_{tick} + C_{act} + C_{sched}$. Supposons maintenant que n soit égal à 20. Lors de son exécution, la tâche τ_1 va prendre la ressource, activer les 19 autres tâches puis libérer la ressource et se terminer. Nous notons C_{AT} le coût du service *ActivateTask* qui ne doit pas être confondu avec la fonction utilisée par toute alarme pour activer la tâche qui lui est associée et représentée par la charge C_{act}. En effet, nous ne pouvons dire si ces deux fonctions sont les mêmes ou pas. Le service *ActiveTask* n'étant jamais utilisé dans notre étude, nous ne relevons aucune mesure de la charge C_{AT}. Ainsi, sur une période T, lorsque n est égal à 20, la durée d'exécution de la tâche τ_1 est égale à $C_{get} + 19 \times C_{AT} + C_{rel} + C_{term}$. Concernant, les autres tâches, celles-ci se contentent de prendre la ressource, de la libérer puis de se terminer. Finalement, sur une période T, la charge totale de travail pour le processeur, lorsque n vaut 20, est égale à $2 \times C_{tick} + C_{act} + C_{sched} + 19 \times C_{AT} + 20 \times (C_{get} + C_{rel} + C_{term})$. Comme au paragraphe précédent, nous fixons la valeur de T à 100000 *cycles* afin d'être assurés que toutes les charges aient toujours fini de s'exécuter avant la prochaine activation de la tâche τ_1. Autrement dit, la période T vérifie :

$$2 \times C_{tick} + C_{act} + C_{sched} + 19 \times C_{AT} + 20 \times (C_{get} + C_{rel} + C_{term}) < T$$

Le tableau 3.5 récapitule les paramètres de toute tâche τ_i appartenant à l'un des trois ensembles de n tâches devant être testés. Ces trois ensembles de n tâches, tous ordonnancés selon la politique FP, correspondent donc aux trois contextes : préemptif, non préemptif et mixte. Nous rappelons que chacun de ces trois ensembles doit être mis en oeuvre vingt fois pour valider n de 1 à 20.

Ensemble de tâche pour l'ordonnancement :	*Priorité*	*Ordonnancement*
FP en contexte préemptif	$P_i = i - 1$	τ_i est préemptive
FP en contexte non préemptif	$P_i = i - 1$	τ_i est non préemptive
FP en contexte mixte	$P_i = i - 1$	τ_i est préemptive si i est pair et non préemptive autrement

TABLE 3.5 – Paramètres de la tâche $\tau_i \in \{\tau_1, ..., \tau_n\}$ pour chacun des trois ensembles de n tâches activées de façon échelonnée avec $n \in [1, 20]$

3.e.iii Caractérisation des charges OSEK

Dans un premier temps, le paragraphe 3.e.iii.1 dresse le bilan des résultats obtenus pour chacune des charges dues au système d'exploitation. Chaque charge peut ensuite être caractérisée par une fonction présentée au paragraphe 3.e.iii.2.

3.e.iii.1 Résultats des mesures sur les charges dues au système d'exploitation

Le tableau 3.6 récapitule, pour chacune des six charges C_{tick}, C_{act}, C_{sched}, C_{term}, C_{get} et C_{rel}, leurs dépendances par rapport au nombre de tâches de l'application qui varie de 1 à 20, au contexte qui peut être préemptif, non préemptif ou mixte, ainsi qu'à la politique d'ordonnancement à savoir FP ou FIFO. Ce tableau indique également les figures représentant les mesures effectuées pour chaque charge.

Charge	*Contexte préemptif, non préemptif, ou mixte*	*Politique d'ordonnancement FP ou FIFO*	*Nombre de tâches de l'application*	*Voir figure*
C_{tick}	X			3.10
C_{act}			X	3.11
C_{sched}	X			3.12
C_{term}		X	X	3.13
C_{get}		X	X	3.14
C_{rel}	X	X	X	3.15

TABLE 3.6 – Dépendances des charges dues au système d'exploitation

La figure 3.10 dresse le bilan des mesures effectuées sur la charge C_{tick}. Nous observons que la durée C_{tick} dépend essentiellement du contexte préemptif, non préemptif ou mixte. La variation observée, lorsque le nombre de tâches vaut 3, correspond à un cycle machine et ce quelque soit le contexte. Cette variation d'un cycle découle directement du code de notre exécutif OSEK, mais demeure négligeable. Ainsi, pour chaque contexte, afin de rester conforme à notre approche pire cas, nous arrondirons, par la suite, la durée C_{tick} à la valeur maximale observée quand $n = 3$. Nous obtiendrons alors un C_{tick} égal à 155 *cycles* en contexte non préemptif, et à 163 *cycles* autrement. Finalement, dans tous les cas, la charge C_{tick} appartient à l'intervalle $[155; 163]$. A partir de sa valeur minimale, la charge C_{tick} augmente donc, au plus, de 5.16%.

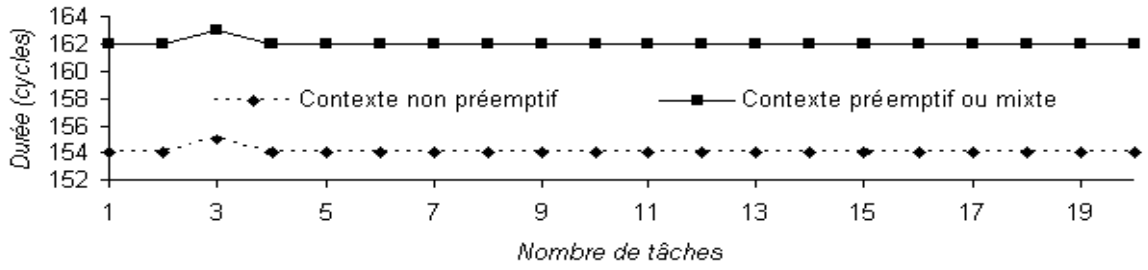

FIGURE 3.10 – Durées pires cas d'exécution de C_{tick}

La synthèse des mesures de la charge C_{act} est exposée à la figure 3.11. Nous constatons que la durée C_{act} dépend exclusivement du nombre de tâches et qu'elle présente une allure logarithmique. Ainsi, nous pouvons penser que, au sein de notre noyau OSEK, la liste des tâches à l'état *prêt* est triée dans l'ordre chronologique selon lequel celles-ci seront exécutées. Autrement dit, lors de son activation, la tâche qui vient de passer à l'état *prêt* serait vraisemblablement insérée, dans une liste triée, à la place qui lui revient selon l'ordonnancement FP/FIFO. Finalement, lorsque la tâche à l'état *courant* se termine, la prochaine tâche élue ne serait autre que la première de la liste précédente. Nous constatons également que l'allure de la durée C_{act} comporte sept paliers : $n = 1$, $n = 2$, $n \in [3, 4]$, $n \in [5, 8]$, $n \in [9, 15]$, $n = 16$ et $n \in [17, 20]$. L'écart maximum entre deux durées, sur un même palier, n'excède jamais 4 *cycles*. En conséquence, nous pourrons arrondir chaque durée à la valeur maximale trouvée sur le palier auquel celle-ci appartient. Les valeurs ainsi obtenues sont spécifiées au paragraphe 3.e.iii.2. Finalement, dans tous les cas, la charge C_{act} appartient à l'intervalle $[316; 631]$. Ainsi, entre deux applications différentes, la valeur de C_{act} peut augmenter de 99.68%.

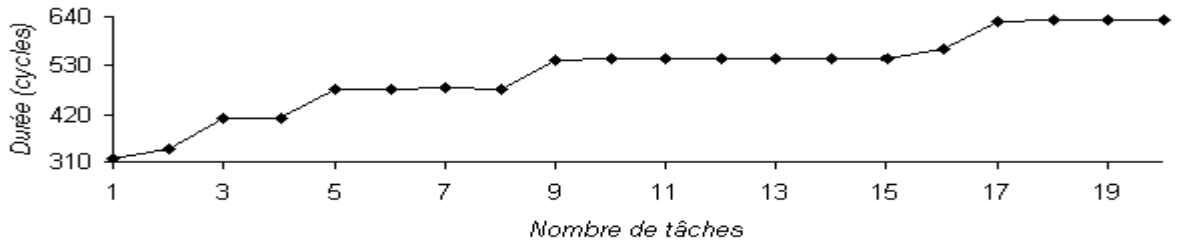

FIGURE 3.11 – Durées pires cas d'exécution de C_{act}

Comme le montre la figure 3.12, la durée de la charge C_{sched} dépend essentiellement du contexte préemptif, non préemptif ou mixte. En contexte non préemptif, nous observons trois variations de la durée C_{sched} lorsque $n = 3$, $n = 9$ et $n = 13$. Dans ce cas, la durée C_{sched} est comprise entre 130 et 134 *cycles*. Ainsi, pour le contexte non préemptif, toutes les durées C_{sched} seront arrondies à 134 *cycles*. Autrement, les contextes préemptif et mixte mènent chacun à une durée constante valant respectivement 126 et 139 *cycles*. Finalement, dans tous les cas, la charge C_{sched} appartient à l'intervalle $[126; 139]$. A partir de sa valeur minimale, la charge C_{sched} augmente donc, au plus, de 10.32%.

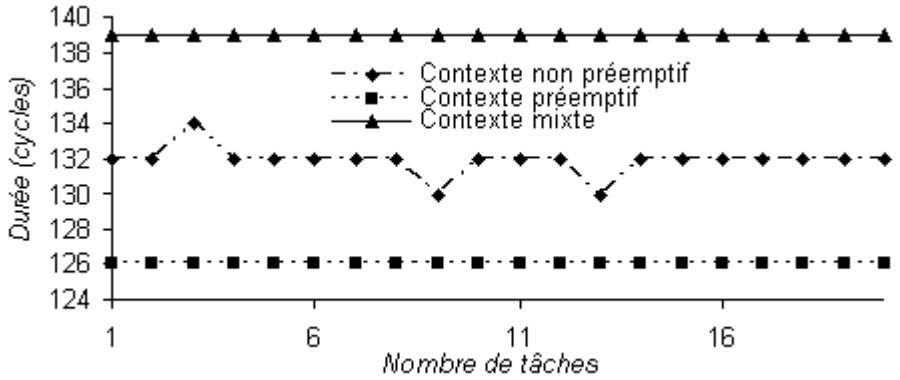

FIGURE 3.12 – Durées pires cas d'exécution de C_{sched}

Nous remarquons, à la figure 3.13, que la durée C_{term} dépend de la politique d'ordonnancement ainsi que du nombre de tâches. Avec l'ordonnancement FIFO, nous observons trois paliers : $n = 1$, $n \in [2, 16]$ et $n \in [17, 20]$. L'ordonnancement FP comporte également trois paliers : $n = 1$, $n \in [2, 15]$ et $n \in [16, 20]$. Nous précisons que l'écart maximal, entre deux valeurs de C_{term} prises sur un même palier pour une politique d'ordonnancement donnée, n'excède jamais 1.43%. En conséquence, nous arrondirons chaque valeur de C_{term} à la valeur maximale observée sur le palier lui correspondant. Les valeurs de C_{term} obtenues, pour chacun de ces six cas, sont précisées au paragraphe 3.e.iii.2. Finalement, dans tous les cas, la charge C_{term} appartient à l'intervalle $[272; 407]$. Ainsi, entre deux applications différentes, la valeur de C_{term} peut augmenter de 49.63%.

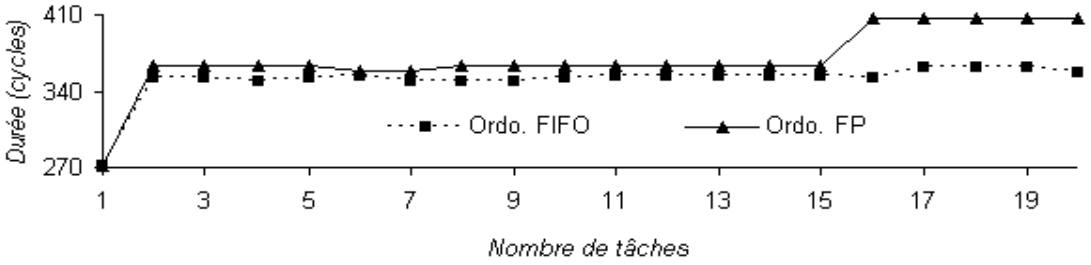

FIGURE 3.13 – Durées pires cas d'exécution de C_{term}

La figure 3.14 montre que la durée C_{get} dépend du nombre de tâche et de la politique d'ordonnancement. Cette durée vaut 183 *cycles* pour $n \in [1, 14]$ si la politique d'ordonnancement est FP et pour $n \in [1, 16]$ autrement. Puis, dans tous les cas, la durée C_{get} passe à 213 *cycles* ce qui revient à une croissance de 16.39%.

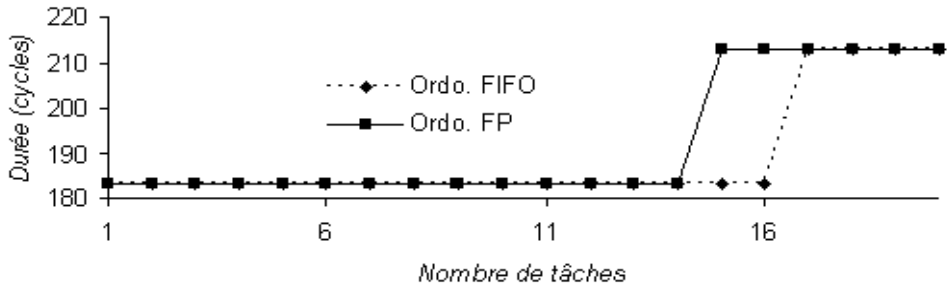

FIGURE 3.14 – Durées pires cas d'exécution de C_{get}

Enfin, la figure 3.15 récapitule les mesures effectuées sur la charge C_{rel}. Nous remarquons que cette durée C_{rel} dépend du nombre de tâches, de la politique d'ordonnancement et du contexte préemptif, non préemptif ou mixte. La durée C_{rel} la plus faible est obtenue en contexte non préemptif. Dans ce cas, celle-ci vaut 138 *cycles* pour $n \in [1, 14]$ ou $n \in [1, 16]$ selon que la politique d'ordonnancement soit FP ou FIFO, puis C_{rel} passe à 164 *cycles*. Dans le cas d'un ordonnancement FIFO en contexte préemptif ou mixte, la durée C_{rel} vaut 225 *cycles* quand $n \in [1, 16]$ puis passe à 260 *cycles*. Dans le cas d'un ordonnancement FP en contexte préemptif ou mixte, la durée C_{rel} vaut 225 *cycles* si $n = 1$, 395 *cycles* si $n \in [2, 15]$, 431 *cycles* si $n = 16$, et 450 *cycles* au delà. Finalement, dans tous les cas, la charge C_{rel} appartient à l'intervalle $[138; 450]$. Ainsi, entre deux applications différentes, la valeur de C_{rel} peut augmenter de 226.09%.

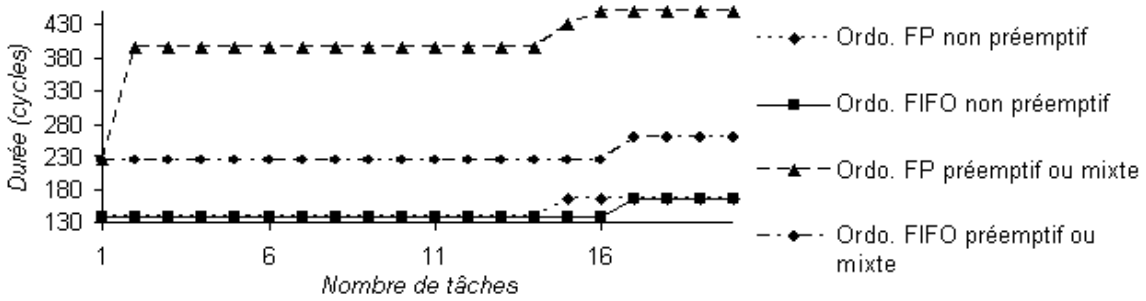

FIGURE 3.15 – Durées pires cas d'exécution de C_{rel}

3.e.iii.2 Fonctions de caractérisation

Les résultats précédents nous permettent d'écrire pour chaque charge due au système d'exploitation une fonction qui la caractérise. Le tableau 3.7 détaille les notations utilisées dans l'écriture de ces fonctions.

Symbole	*Description*
P	Contexte préemptif
NP	Contexte non préemptif
M	Contexte mixte
F	Ordonnancement FIFO
NF	Ordonnancement FP
n	Nombre de tâches de l'application

TABLE 3.7 – Notations pour les fonctions de caractérisation

Nous rappelons que, conformément à ce qui est dit au paragraphe 3.e.iii.1, les valeurs initiales ayant été arrondies, l'ont toujours été à une valeur supérieure. Ainsi, nous obtenons pour chaque charge des bornes supérieures qui mènent bien à un calcul pire cas des temps de réponse des tâches de l'application. Les fonctions de caractérisation finales, exprimées en *cycles*, sont exposées ci-après :

$$C_{tick} = \begin{cases} 155 & \text{si NP et } n \leq 20 \\ 163 & \text{si (P or M) et } n \leq 20 \end{cases}$$

$$C_{sched} = \begin{cases} 126 & \text{si P} \\ 134 & \text{si NP} \\ 139 & \text{si M} \end{cases}$$

$$C_{act} = \begin{cases} 316 & \text{si } n = 1 \\ 339 & \text{si } n = 2 \\ 412 & \text{si } n \in [3,4] \\ 480 & \text{si } n \in [5,8] \\ 546 & \text{si } n \in [9,15] \\ 566 & \text{si } n = 16 \\ 631 & \text{si } n \in [17,20] \end{cases}$$

$$C_{term} = \begin{cases} 272 & \text{si } n = 1 \\ 354 & \text{si F et } n \in [2,16] \\ 363 & \text{si F et } n \in [17,20] \\ 364 & \text{si NF et } n \in [2,15] \\ 407 & \text{si NF et } n \in [16,20] \end{cases}$$

$$C_{get} = \begin{cases} 183 & \text{si F et } n \in [1,16] \\ 183 & \text{si NF et } n \in [1,14] \\ 213 & \text{si F et } n \in [17,20] \\ 213 & \text{si NF et } n \in [15,20] \end{cases}$$

$$C_{rel} = \begin{cases} 138 & \text{si NF et NP et } n \in [1,14] \\ 164 & \text{si NF et NP et } n \in [15,20] \\ 138 & \text{si F et NP et } n \in [1,16] \\ 164 & \text{si F et NP et } n \in [17,20] \\ 225 & \text{si (P ou M) et } n = 1 \\ 395 & \text{si NF et (P ou M) et } n \in [2,14] \\ 431 & \text{si NF et (P ou M) et } n = 15 \\ 450 & \text{si NF et (P ou M) et } n \in [16,20] \\ 225 & \text{si F et (P ou M) et } n \in [2,16] \\ 260 & \text{si F et (P ou M) et } n \in [17,20] \end{cases}$$

3.f Synthèse

Tout système d'exploitation temps réel, conforme au standard OSEK, est obligatoirement et totalement statique. Autrement dit, tous les objets, que ce soient les tâches, les alarmes, ou encore les ressources, qui constituent l'application, doivent être définis, une fois pour toute, lors de sa conception. Aucun nouvel objet ne pourra être créé dynamiquement durant l'exécution de l'application. De même, chaque tâche se voit attribuer, lors de la conception, une priorité qui ne peut en aucun cas être modifiée par la suite. Le standard OSEK met en oeuvre la politique FP/FIFO pour l'ordonnancement qui peut être préemptif, non préemptif ou mixte. De plus, l'accès à chaque ressource est protégé par la méthode du plafond de priorité qui a pour principal avantage d'éviter tout risque d'interblocage. Enfin, les alarmes offrent un support idéal pour l'implémentation de tâches périodiques. En vue du chapitre 4, nous avons identifié et mesuré les différentes charges dues au noyau OSEK, nécessaires à la gestion de la base de temps ainsi qu'à celles des tâches et du mécanisme du plafond de priorité, afin de pouvoir les intégrer aux conditions de faisabilité classiques.

Chapitre 4

Conditions de faisabilité avec l'ordonnancement FP/FIFO

Ce chapitre présente notre analyse temps réel pour un système monoprocesseur basé sur une implémentation du standard OSEK fournie par la société Vector et conforme à la spécification 2.2 de ce standard [49]. Nous rappelons que les principes de ce standard sont explicités au chapitre 3. Les différentes charges, dues au système d'exploitation, y sont également identifiées et caractérisées. Notre objectif consiste, pour une application temps réel quelconque, composée de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes et non concrètes, à calculer précisément le temps de réponse pire cas de chaque tâche, ainsi que le facteur d'utilisation du processeur. Nous précisons que notre étude ne s'applique qu'à des ensembles de tâches se partageant, au plus, une ressource. A la section 4.a, nous exposons d'autres travaux de recherche visant à intégrer le coût du système d'exploitation dans les conditions de faisabilité et positionnons notre approche par rapport à ceux-ci. Puis, à la section 4.b, nous illustrons l'impact des charges, exposées à la section 3.e, sur le calcul du temps de réponse pire cas d'une tâche. Dans cette même section, nous expliquons comment prendre en compte toute inversion de priorité due à un effet non préemptif ou au mécanisme du plafond de priorité. Enfin, nous présentons, à la section 4.c, nos conditions de faisabilité tenant compte des charges dues au système d'exploitation OSEK. De plus, nous expliquons, à la section 4.d, comment considérer les tâches sporadiques et les interruptions. La section 4.e, quant à elle, résume la contribution de ce chapitre à l'amélioration des outils de dimensionnement temps réel actuels.

4.a Etat de l'art sur la prise en compte du coût du système d'exploitation

Les conditions de faisabilité exposées aux sections 2.d et 2.e négligent l'exécution du système d'exploitation ce qui conduit à une sous estimation des temps de réponse pires cas. Cette lacune peut conduire à valider des systèmes temps réels qui, une fois déployés, ne parviennent pas à respecter leurs contraintes temporelles. Pour éviter ce problème, la plupart du temps, les concepteurs n'utilisent pas pleinement le processeur. Cette marge de sécurité, difficile à estimer, peut alors devenir excessive et lourde de conséquences économiques sur des productions à grande échelle. Notre objectif étant de combler cette lacune, cette section présente les principaux travaux menés dans ce sens. La sous section 4.a.i présente une étude visant à déterminer le nombre exact de préemptions que subit chaque tâche du système afin d'en intégrer précisément le coût dans les conditions de faisabilité. Les sous sections 4.a.ii et 4.a.iii traitent respectivement d'ordonnancements guidés par le temps et par les événements. Enfin, la sous section 4.a.iv présente les travaux menés sur le standard OSEK.

4.a.i Détermination du nombre exact de préemptions

Dans le papier [41], P.Meumeu Yomsi et Y.Sorel considèrent un ensemble de n tâches à échéances sur requête, périodiques, indépendantes, préemptives et concrètes. Les priorités sont fixes et déterminées à l'aide de l'algorithme hors ligne RM détaillé au paragraphe 2.b.i.1. Les auteurs proposent un algorithme permettant de déterminer le nombre exact de préemptions que subissent les différentes instances de chaque tâche. La figure 4.1 illustre les préemptions, ayant chacune une durée d'exécution égale à 1, sur un ensemble de deux tâches τ_1 et τ_2 telles que $C_1 = 2$, $C_2 = 3$, $T_1 = 5$ et $T_2 = 8$.

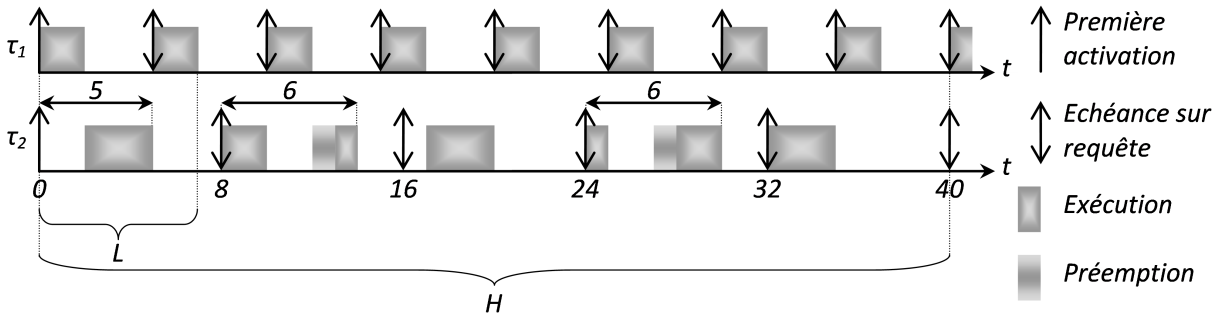

FIGURE 4.1 – Illustration de l'algorithme proposé par P.Meumeu Yomsi et Y.Sorel

Comme le montre la figure 4.1, nous constatons que le temps de réponse pire cas de la tâche τ_2 n'intervient pas durant la période d'activité notée L bien que le scénario d'activation synchrone ait été appliqué. En effet, une préemption ne se produit que lorsqu'une tâche en cours d'exécution est interrompue par une tâche plus prioritaire. Dans l'exemple ci-dessus, étant donné que $T_1 < T_2$, selon l'ordonnancement RM, la tâche τ_1 est plus prioritaire que la tâche τ_2. Nous observons alors deux préemptions : lors de la seconde puis de la quatrième instance de la tâche τ_2. Ainsi, la recherche des temps de réponse pires cas ne se limite pas

à l'étude de la période d'activité L mais à l'hyperpériode notée H. Etant donné que toutes les tâches présentes sont périodiques, indépendamment du scénario d'activation, le motif des activations se répète également de façon périodique. L'**hyperpériode** correspond à la période de ce motif et est égale au plus petit multiple commun aux périodes de l'ensemble des tâches du système. Dans le cas présent, nous avons donc $H = ppcm(5, 8) = 40$. L'algorithme proposé permet de déterminer toutes les préemptions qui ont lieu dans cette hyperpériode puis de les intégrer au temps de réponse pire cas de chaque tâche ainsi qu'au facteur d'utilisation du processeur. L'algorithme traite les tâches, une par une, de la plus à la moins prioritaire. Nous soulignons que cet algorithme n'est applicable qu'avec des tâches concrètes, c'est à dire des tâches dont nous connaissons les dates d'activation.

Dans le papier [42], P.Meumeu Yomsi et Y.Sorel dénombrent également le nombre de préemptions, cette fois, dans un système de tâches à périodicité stricte liées par des contraintes de précédence. Par périodicité stricte, nous entendons que l'exécution de n'importe quelle tâche doit débuter dès que celle-ci est activée. Comme précédemment, les priorités sont fixes et déterminées selon l'algorithme RM.

4.a.ii Prise en compte du coût d'un OS guidé par le temps

Dans l'article [15], A.Burns, K.Tindell et A.Wellings considèrent un ensemble noté τ comportant n tâches préemptives, périodiques ou sporadiques, non concrètes et telles que, $\forall \tau_i \in \tau$, $D_i \leq T_i$. Les priorités sont fixes et déterminées à l'aide de l'algorithme hors ligne DM présenté au paragraphe 2.b.i.2. De plus, les auteurs considèrent le cas où les différentes tâches partagent une ressource dont l'accès est protégé à l'aide du mécanisme du plafond de priorité (voir sous section 3.c.ii). L'ordonnancement étant guidé par le temps, les auteurs mettent en évidence deux charges dues au système d'exploitation :

- C_{ql} : Cette charge équivaut à C_{act} et correspond au coût de l'activation d'une tâche.
- C_{int} : Cette charge équivaut à C_{tick} et correspond à une interruption périodique. Durant cette interruption, le système d'exploitation gère sa base de temps et, si une ou plusieurs tâches plus prioritaires que la tâche courante ont été activées depuis l'interruption précédente, attribue le processeur à la tâche la plus prioritaire parmi celles-ci. Ainsi, lorsqu'une tâche est activée, son exécution ne débute qu'après la prochaine interruption.

Dans ce papier, les auteurs proposent une équation permettant de tenir compte des deux charges précédentes lors du calcul des temps de réponse pires cas obtenus en appliquant le scénario pire cas décrit au paragraphe 2.c.ii.1.

4.a.iii Prise en compte du coût d'un exécutif guidé par les événements

Dans le papier [2], C.K.Angelov, I.E.Ivanov, et I.J.Haratcherev, étudient un exécutif basé sur un ordonnancement à priorités fixes guidé par les événements. Les auteurs s'intéressent à un système constitué de tâches périodiques et d'interruptions externes assimilées à des tâches sporadiques. Dans tous les cas, toute tâche τ_i de ce système est préemptive, non concrète et vérifie $D_i \leq T_i$. De plus, les différentes tâches partagent une ressource

dont l'accès est protégé à l'aide du mécanisme du plafond de priorité (voir sous section 3.c.ii). Quatre charges dues au système d'exploitation sont intégrées au calcul du temps de réponse pire cas :

- C_{clock} : Cette charge équivaut à C_{tick} et correspond à une interruption périodique nécessaire au système d'exploitation pour gérer sa base de temps.
- C_{ptr} : Cette charge équivaut à C_{act} et représente le coût de l'activation d'une tâche périodique.
- C_{pro} : Sachant que le système d'exploitation employé met en oeuvre un ordonnancement à double priorité, c'est à dire que chaque tâche est activée avec une première priorité fixe puis passe à une nouvelle priorité fixe plus élevée au bout d'un certain temps, cette charge représente le coût de cette promotion.
- C_{isr}^{j} : Les auteurs ayant assimilé chaque interruption à une tâche sporadique, cette charge correspond à l'exécution de la fonction d'interruption chargée d'activer la tâche sporadique τ_j.

Dans ce papier, les auteurs proposent une équation permettant d'intégrer les quatre charges précédentes lors du calcul des temps de réponse pires cas obtenus en suivant le scénario pire cas décrit au paragraphe 2.c.ii.1.

4.a.iv Prise en compte du coût d'un système d'exploitation OSEK

Dans le papier [51], après un rappel de la norme OSEK et une description de leurs outils, S.H.Seo, S.W.Lee, S.H.Hwang et J.W.Jeon énumèrent une liste de charges dues au système d'exploitation puis indiquent la valeur de chacune d'entre elles pour une application donnée. Malheureusement, ce papier n'explique pas comment intégrer ces charges aux conditions de faisabilité.

Les travaux effectués tout au long de cette thèse, afin d'identifier et d'intégrer aux conditions de faisabilité les différentes charges dues à un noyau OSEK, ont donné lieu à plusieurs publications. Dans l'article [11], après avoir identifié les charges C_{tick}, C_{act}, C_{sched} et C_{term}, nous montrons comment en tenir compte dans le cas d'un ensemble τ comportant n tâches préemptives, périodiques, indépendantes, non concrètes et telles que, $\forall \tau_i \in \tau$, $D_i \leq T_i$. Cette première étude, portant sur l'ordonnancement FP préemptif, a ensuite été élargie à l'ordonnancement FP/FIFO préemptif à l'occasion du papier [10], puis à l'ordonnancement FP/FIFO mixte au papier [12], à chaque fois dans le cas de tâches à échéance arbitraire. Les charges C_{get} et C_{rel}, liées au mécanisme du plafond de priorité, ont ensuite été mises en avant et intégrées à nos conditions de faisabilité lors du papier [9]. De plus, comme nous l'expliquons à la section 4.d, selon notre approche pire cas, une tâche sporadique est assimilée à une tâche périodique. Finalement, notre étude est valable pour l'ordonnancement FP/FIFO de tout système doté, au plus, d'une ressource et d'un ensemble de tâches à échéance arbitraire, périodiques ou sporadiques, préemptives ou non, indépendantes et non concrètes. Nous soulignons que nos résultats sont plus généraux que les autres travaux présentés dans cette section. En effet, l'étude présentée à la sous section 4.a.i n'est applicable qu'avec des tâches concrètes. Quant aux travaux présentés aux sous sections 4.a.ii et 4.a.iii, ceux-ci sont limités à l'ordonnancement FP préemptif d'un ensemble de tâches τ tel que, $\forall \tau_i \in \tau$, $D_i \leq T_i$.

4.b Illustration du problème

A la sous section 4.b.i, nous montrons comment les charges dues au système d'exploitation s'intègrent dans le calcul du temps de réponse d'une tâche. A cette occasion, nous commençons, à l'aide d'un exemple simple basé sur une seule tâche, par préciser le modèle temporel de nos tâches ainsi que la définition de leur temps de réponse. Puis, sur un exemple plus complexe basé sur un ensemble de quatre tâches, nous calculons le temps de réponse d'une tâche dont l'échéance est inférieure à la période. Suite à l'exemple précédent, à la sous section 4.b.ii, nous mettons en avant le problème posé par la prise en compte de la charge C_{sched} dans la détermination des temps de réponse pires cas et proposons une solution à celui-ci. A la sous section 4.b.iii, nous revenons sur les inversions de priorité dues à l'effet non préemptif et au mécanisme du plafond de priorité. Enfin, à la sous section 4.b.iv, nous récapitulons les règles à observer, dans un cas général, pour intégrer les charges, dues au système d'exploitation OSEK, aux conditions de faisabilité.

4.b.i Calcul du temps de réponse d'une tâche préemptive ayant une échéance inférieure à sa période

Nous revenons, au paragraphe 4.b.i.1, sur le modèle temporel de nos tâches ainsi que sur la définition de leur temps de réponse. Puis, au paragraphe 4.b.i.2, nous procédons au calcul du temps de réponse d'une tâche appartenant à un ensemble de quatre tâches périodiques préemptives.

4.b.i.1 Modèle temporel et temps de réponse des tâches

Comme nous l'avons déjà annoncé au cours des chapitres précédents, nous ne nous intéressons, dans cette thèse, qu'aux ensembles de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes et non concrètes. Ainsi, comme nous l'avons défini à la sous section 2.a.i, chaque tâche τ_i est caractérisée, entre autres, par sa période, sa durée d'exécution pire cas et son temps de réponse pire cas respectivement notés T_i, C_i et r_i. La figure 4.2 illustre l'exécution d'une tâche seule accédant à une ressource :

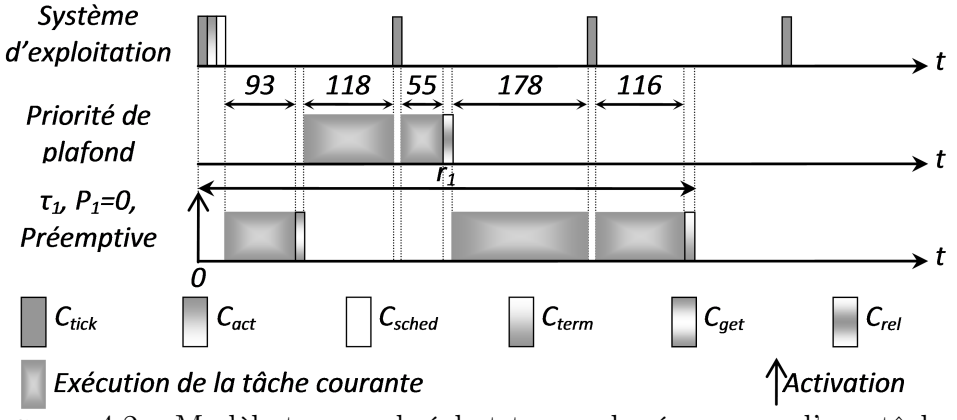

FIGURE 4.2 – Modèle temporel réel et temps de réponse r_1 d'une tâche τ_1

Comme nous l'observons à la figure 4.2, la première activation de la tâche préemptive τ_1 a lieu à la date 0. A cette date, au niveau du système d'exploitation, nous observons une charge C_{tick} nécessaire à la gestion de sa base de temps ainsi qu'à celle de l'alarme en charge des activations de la tâche τ_1. Cette première charge C_{tick} est suivie d'une charge C_{act} qui concrétise l'activation de la tâche τ_1 puis d'une charge C_{sched} qui effectue le changement de contexte nécessaire à son exécution. Après cela, la tâche τ_1 s'exécute pendant 93 *cycles* puis appelle le service *GetResource* correspondant à la charge C_{get}. Une fois la ressource acquise, la tâche τ_1 poursuit son exécution pendant 118 *cycles* avant d'être interrompue par une charge C_{tick}. Après cela, la tâche τ_1 s'exécute encore durant 55 *cycles* avant de libérer la ressource grâce au service *ReleaseResource* matérialisé par la charge C_{rel}. Dès que la ressource est libérée, l'exécution de la tâche τ_1 reprend pendant 178 *cycles* avant que celle-ci ne soit à nouveau interrompue par une charge C_{tick}. Enfin, son exécution se poursuit sur 116 *cycles* au bout desquels la tâche τ_1 appelle le service *TerminateTask* dont l'exécution est représentée par la charge C_{term}.

Lors de la conception d'une tâche, son **temps d'exécution pire cas** doit être mesuré indépendamment de toute charge liée au système d'exploitation ou à ses services. Autrement dit, le concepteur ne mesure que le temps d'exécution pire cas de son propre code. En supposant qu'à la figure 4.2, le temps d'exécution de la tâche τ_1 ait été maximum, son temps d'exécution pire cas n'intègre aucune des six charges liées au système d'exploitation ou à ses services et vaut $C_1 = 93 + 118 + 55 + 178 + 116 = 560$ *cycles*.

Jusqu'à présent, nous définissions le temps de réponse d'une tâche comme étant la différence entre sa date de fin d'exécution et sa date d'activation. Considérant maintenant les charges dues au système d'exploitation, nous devons préciser cette définition : le **temps de réponse** d'une tâche correspond à la différence entre la date où la charge C_{term}, clôturant son exécution, se termine et sa date d'activation. Ainsi, comme l'illustre la figure 4.2, le temps de réponse r_1 de la tâche τ_1 vaut $r_1 = C_1 + 3 \times C_{tick} + C_{act} + C_{sched} + C_{get} + C_{rel} + C_{term}$.

Nous soulignons que l'activation d'une tâche n'étant en réalité jamais effective à la date demandée mais après l'exécution de la charge C_{act} correspondante, bien que nous parlons d'**activation** et continuerons ainsi par souci pédagogique, certains pourraient préférer parler de **demande d'activation**.

Sachant que nous ne nous intéressons, dans cette thèse, qu'à des ensembles de tâches ne comportant jamais plus d'une ressource, comme nous l'évoquons à la sous section 4.a.ii, chaque tâche τ_i est également caractérisée par un paramètre B_i représentant la durée pendant laquelle celle-ci utilise la ressource. Tout comme lors de la mesure de son paramètre C_i, le concepteur évalue le paramètre B_i de la tâche τ_i sans tenir compte des charges liées au système d'exploitation ou à ses services. Ainsi, nous constatons à la figure 4.2 que la tâche τ_1 monopolise la ressource pendant une durée B_1 égale à $118 + 55 = 173$ *cycles*. Nous soulignons que la durée B_i d'une tâche τ_i est incluse dans sa durée C_i.

4.b.i.2 Calcul du temps de réponse d'une tâche préemptive ayant une échéance inférieure à sa période

Nous proposons, dans ce paragraphe, sur un exemple simple, de calculer le temps de réponse d'une tâche en tenant compte des charges dues au système d'exploitation. Cette illustration vise à montrer clairement comment chacune de ces charges, en fonction de son comportement, sera intégrée dans les conditions de faisabilité présentées à la section 4.c. Nous allons donc calculer le temps de réponse de la tâche τ_2 appartenant à l'ensemble de quatre tâches préemptives τ défini au tableau 4.1. Nous supposons que cet ensemble est ordonnancé avec une valeur T_{tick} égale à 127 *cycles*.

Tâche	C $(cycles)$	B $(cycles)$	D $(cycles)$	T $(cycles)$	P
τ_4	38	0	381	508	3
τ_3	50	10	381	508	2
τ_2	472	100	1150	1270	1
τ_1	150	0	1500	889	0

TABLE 4.1 – Ensemble de tâches pour l'illustration d'un calcul de temps de réponse

Comme nous le constatons au tableau 4.1, seules les tâches τ_2 et τ_3 utilisent la ressource. La tâche τ_4 étant plus prioritaire que n'importe quelle autre tâche, sa priorité sera donc supérieure à la priorité de plafond associée à la ressource. La figure 4.3 illustre l'ordonnancement de cet ensemble de tâches lorsque celles-ci sont activées simultanément à la date 0.

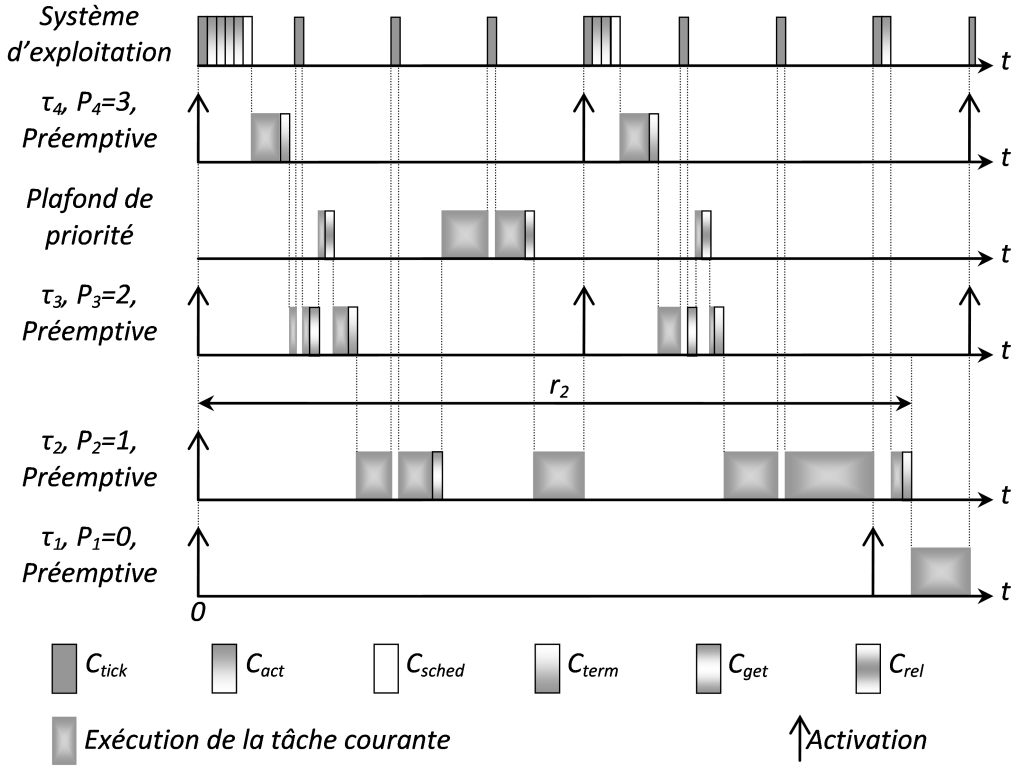

FIGURE 4.3 – Calcul d'un temps de réponse avec activations simultanées

Le rôle des différentes charges du système d'exploitation ayant déjà été largement détaillé à la sous section 3.e.i, nous ne détaillons pas ici l'ordonnancement présenté à la figure 4.3 et nous contentons d'étudier le temps de réponse r_2 de la tâche τ_2. Comme nous l'observons à la figure 4.3, ce temps de réponse vaut :

$$r_2 = 8 \times C_{tick} + 7 \times C_{act} + 2 \times C_{sched} + 5 \times C_{term} + 3 \times C_{get} + 3 \times C_{rel} + 2 \times C_4 + 2 \times C_3 + C_2$$

Supposons, pour cet exemple, que chacune des six charges du système d'exploitation vaille 12 $cycles$, nous aurions alors $r_2 = 984$ $cycles$.

Nous allons maintenant retrouver ce résultat par le calcul. Premièrement, du fait que les quatre tâches sont activées simultanément, ce scénario est conforme au scénario pire cas exposé au paragraphe 2.c.ii.1. De plus, étant donné que $D_2 < T_2$, comme nous l'expliquons à la sous section 2.d.i, le temps de réponse précédent devrait être le temps de réponse pire cas de la tâche τ_2. Nous verrons, à la sous section 4.b.ii, que dans le cas réel, ce temps de réponse r_2 ne correspond pas au pire cas de la tâche τ_2. Toutefois, repartons de la formule donnée à la sous section 2.d.i en contexte préemptif :

$$r_2 = \omega_{2,0} \text{ où } \omega_{2,0} \text{ est solution de } \omega_{2,0} = C_2 + \sum_{\tau_j \in hp(2)} \lceil \frac{\omega_{2,0}}{T_j} \rceil C_j$$

Nous précisons que $hp(i)$ correspond à l'ensemble des tâches strictement plus prioritaires que la tâche τ_i. De même, $lp(i)$ désigne l'ensemble des tâches strictement moins prioritaires que la tâche τ_i. Ainsi, dans le cas présent, nous avons $hp(2) = \{\tau_3, \tau_4\}$ et $lp(2) = \{\tau_1\}$. La date $\omega_{2,0}$ correspond, quant à elle, à la date à laquelle l'exécution de la tâche τ_2 activée à la date 0 s'achève.

Avant la date $\omega_{2,0}$, la tâche τ_2 s'exécute naturellement une fois. D'où le terme C_2 dans l'équation précédente. De plus, la tâche τ_2 étant préemptive, tant que son exécution n'est pas terminée, celle-ci est périodiquement interrompue par les tâches appartenant à $hp(2)$. Le calcul de la date $\omega_{2,0}$ comptabilise donc toutes les exécutions des tâches de $hp(2)$ sur l'intervalle $[0, \omega_{2,0}[$. Notez que la date $\omega_{2,0}$ est exclue de l'intervalle précédent car, à cette date, l'exécution de la tâche τ_2 activée à la date 0 est terminée. Aussi, en toute logique, aucune tâche, même appartenant à $hp(2)$, activée à partir de la date $\omega_{2,0}$, ne peut interférer dans l'exécution de la tâche τ_2 activée à la date 0. Voyons maintenant, au cas par cas, comment s'intègrent, dans l'équation précédente, les différentes charges dues au système d'exploitation :

- **Charge C_{tick} :** Comme nous l'avons vu à la sous section 3.e.i, la charge C_{tick} s'exécute périodiquement, avec une période T_{tick} qu'une tâche soit en train de s'exécuter ou non. De plus, lorsqu'une tâche s'exécute, que celle-ci soit préemptive ou non, cette charge la préempte toujours. Ainsi, nous devons comptabiliser, lors du calcul de la date $\omega_{2,0}$, l'ensemble des charges C_{tick} arrivées sur l'intervalle $[0, \omega_{2,0}[$:

  $$\omega_{2,0} = C_2 + \sum_{\tau_j \in hp(2)} \lceil \frac{\omega_{2,0}}{T_j} \rceil C_j + \lceil \frac{\boldsymbol{\omega_{2,0}}}{\mathbf{T_{tick}}} \rceil \mathbf{C_{tick}}$$

  Nous soulignons que les travaux exposés aux sous sections 4.a.ii et 4.a.iii intègrent cette charge de la même façon.

– **Charge C_{act} :** Concernant la tâche τ_2 activée à la date 0, lorsque son alarme survient à cette date, celle-ci s'accompagne d'une charge C_{act} qui s'ajoute donc à sa durée d'exécution pire cas C_2. De même, à chaque fois qu'une tâche τ_j de $hp(2)$ est activée sur l'intervalle $[0, \omega_{2,0}[$, sa durée d'exécution pire cas C_j est elle-aussi augmentée d'une charge C_{act}. Enfin, concernant la seule tâche τ_1 appartenant à $lp(2)$, bien que celle-ci ne s'exécute pas dans l'intervalle $[0, \omega_{2,0}[$, chacune de ses activations dans cet intervalle est tout de même concrétisée par une charge C_{act}. En effet, si tel n'était pas le cas, les activations de la tâche τ_1 dans l'intervalle $[0, \omega_{2,0}[$ seraient alors perdues. Ainsi, nous arrivons à l'équation :
$$\omega_{2,0} = \mathbf{C_{act}} + C_2 + \sum_{\tau_j \in hp(2)} \lceil \frac{\omega_{2,0}}{T_j} \rceil (\mathbf{C_{act}} + C_j) + \lceil \frac{\boldsymbol{\omega_{2,0}}}{\mathbf{T_1}} \rceil \mathbf{C_{act}} + \lceil \frac{\omega_{2,0}}{T_{tick}} \rceil C_{tick}$$
Finalement, une charge C_{act} est comptabilisée pour toute activation, de n'importe quelle tâche de τ, sur l'intervalle $[0, \omega_{2,0}[$. Aussi, les travaux exposés aux sous sections 4.a.ii et 4.a.iii intègrent cette charge de façon équivalente.
– **Charge C_{term} :** La prise en compte de cette charge est d'autant plus simple que celle-ci intervient systématiquement à la fin de chaque exécution. Ainsi, dans l'équation précédente, une charge C_{term} doit être ajoutée à chaque durée d'exécution pire cas. Nous arrivons alors à :
$$\omega_{2,0} = C_{act} + C_2 + \mathbf{C_{term}} + \sum_{\tau_j \in hp(2)} \lceil \frac{\omega_{2,0}}{T_j} \rceil (C_{act} + C_j + \mathbf{C_{term}}) + \lceil \frac{\omega_{2,0}}{T_1} \rceil C_{act} + \lceil \frac{\omega_{2,0}}{T_{tick}} \rceil C_{tick}$$
Nous soulignons que les travaux exposés aux sous sections 4.a.ii et 4.a.iii ne font apparaître aucune charge équivalente à C_{term}.
– **Charge C_{sched} :** Du fait que cette charge apparaisse à chaque préemption, son intégration n'est pas aisée. En effet, à l'heure actuelle, à notre connaissance, aucune méthode n'existe pour déterminer le nombre maximal de préemptions que subit chaque tâche d'un ensemble de tâches non concrètes ordonnancé FP/FIFO. Pour l'instant, nous nous contentons du fait que, dans le cas présent, comme nous l'observons à la figure 4.3, cette charge apparaît périodiquement avec une période égale à T_4. Nous ajoutons ainsi à l'équation précédente la charge de travail nécessitée par la charge C_{sched} sur l'intervalle $[0, \omega_{2,0}[$:
$$\omega_{2,0} = C_{act} + C_2 + C_{term} + \sum_{\tau_j \in hp(2)} \lceil \frac{\omega_{2,0}}{T_j} \rceil (C_{act} + C_j + C_{term}) + \lceil \frac{\omega_{2,0}}{T_1} \rceil C_{act} + \lceil \frac{\omega_{2,0}}{T_{tick}} \rceil C_{tick} + \lceil \frac{\boldsymbol{\omega_{2,0}}}{\mathbf{T_4}} \rceil \mathbf{C_{sched}}$$
Là encore, les travaux exposés aux sous sections 4.a.ii et 4.a.iii ne font apparaître aucune charge équivalente à C_{sched}.
– **Charges C_{get} et C_{rel} :** Ces deux charges interviennent chacune une fois durant chaque exécution des tâches τ_2 et τ_3. En conséquence, nous devons les compter toutes deux une fois pour l'exécution de la tâche τ_2 puis autant de fois que la tâche τ_3 est activée sur l'intervalle $[0, \omega_{2,0}[$:
$$\omega_{2,0} = C_{act} + C_2 + \mathbf{C_{get}} + \mathbf{C_{rel}} + C_{term} + \sum_{\tau_j \in hp(2)} \lceil \frac{\omega_{2,0}}{T_j} \rceil (C_{act} + C_j + C_{term}) + \lceil \frac{\boldsymbol{\omega_{2,0}}}{\mathbf{T_3}} \rceil (\mathbf{C_{get}} + \mathbf{C_{rel}}) + \lceil \frac{\omega_{2,0}}{T_1} \rceil C_{act} + \lceil \frac{\omega_{2,0}}{T_{tick}} \rceil C_{tick} + \lceil \frac{\omega_{2,0}}{T_4} \rceil C_{sched}$$
Aucune charge similaire n'apparaît dans les travaux exposés aux sous sections 4.a.ii et 4.a.iii.

L'équation précédente définit une suite dont nous déterminons maintenant le premier terme $\omega_{2,0}^0$. Celui-ci comptabilise l'ensemble des exécutions prises en compte dans le calcul de $\omega_{2,0}$ et découlant de la seule activation simultanée des quatre tâches à la date 0, c'est à dire :

- les activations et les exécutions des tâches τ_2, τ_3 et τ_4 qui amènent une charge de travail égale à :
  $(C_{act}+C_4+C_{term})+(C_{act}+C_3+C_{get}+C_{rel}+C_{term})+(C_{act}+C_2+C_{get}+C_{rel}+C_{term})$
  $= 3 \times C_{act} + 3 \times C_{term} + 2 \times (C_{get} + C_{rel}) + C_4 + C_3 + C_2$
- l'activation de la tâche τ_1 qui revient à une charge C_{act}
- les charges C_{tick} et C_{sched} qui précèdent l'exécution de la tâche τ_4.

Ainsi, nous obtenons :

$$\omega_{2,0}^0 = C_{tick} + 4 \times C_{act} + C_{sched} + 3 \times C_{term} + 2 \times (C_{get} + C_{rel}) + C_4 + C_3 + C_2$$

Finalement, la date $\omega_{2,0}$ se résout par la détermination du plus petit point fixe de la suite :

$$\begin{cases} \omega_{2,0}^0 = C_{tick} + 4 \times C_{act} + C_{sched} + 3 \times C_{term} + 2 \times (C_{get} + C_{rel}) + C_4 + C_3 + C_2 \\ \omega_{2,0}^{m+1} = C_{act} + C_2 + C_{get} + C_{rel} + C_{term} + \sum_{\tau_j \in hp(2)} \lceil \frac{\omega_{2,0}^m}{T_j} \rceil (C_{act} + C_j + C_{term}) + \\ \lceil \frac{\omega_{2,0}^m}{T_3} \rceil (C_{get} + C_{rel}) + \lceil \frac{\omega_{2,0}^m}{T_1} \rceil C_{act} + \lceil \frac{\omega_{2,0}^m}{T_{tick}} \rceil C_{tick} + \lceil \frac{\omega_{2,0}^m}{T_4} \rceil C_{sched} \end{cases}$$

Passons maintenant à l'application numérique :

$$\begin{aligned} \omega_{2,0}^0 &= 716 \\ \omega_{2,0}^1 &= 520 + \lceil \tfrac{716}{508} \rceil 172 + \lceil \tfrac{716}{889} \rceil 12 + \lceil \tfrac{716}{127} \rceil 12 = 948 \\ \omega_{2,0}^2 &= 520 + \lceil \tfrac{948}{508} \rceil 172 + \lceil \tfrac{948}{889} \rceil 12 + \lceil \tfrac{948}{127} \rceil 12 = 984 \\ \omega_{2,0}^3 &= 520 + \lceil \tfrac{984}{508} \rceil 172 + \lceil \tfrac{984}{889} \rceil 12 + \lceil \tfrac{984}{127} \rceil 12 = 984 = \omega_{2,0}^2 \end{aligned}$$

Comme nous l'avons observé à la figure 4.3, le temps de réponse r_2 de la tâche τ_2, activée à la date 0, vaut bien 984 *cycles*.

A la section 4.c, nous étudierons les conditions de faisabilité générales, pour l'ordonnancement FP/FIFO, en y intégrant les charges dues au système d'exploitation comme nous venons de le faire dans l'exemple précédent. Toutefois, comme nous le verrons à la sous section 4.b.ii, lorsque toutes les tâches ne sont pas activées simultanément à la date 0, la tâche τ_2 peut subir davantage de préemptions et donc obtenir un temps de réponse supérieur au précédent. Aussi, nous montrons, à la sous section 4.b.ii, comment intégrer correctement la charge C_{sched} afin d'être assurés que le temps de réponse de la tâche τ_2, obtenu en applicant le scénario pire cas décrit au paragraphe 2.c.ii.1, constitue bien une borne supérieure à tout temps de réponse de cette tâche.

4.b.ii Problème d'intégration de la charge C_{sched} dans les calculs des temps de réponse pires cas

Revenons sur l'ensemble de quatre tâches préemptives indépendantes τ dont les caractéristiques sont détaillées au tableau 4.1. Au paragraphe 4.b.i.2, nous avons calculé le temps de réponse r_2 de la tâche τ_2 appartenant à cet ensemble. Nous avons alors expliqué que, l'ensemble des tâches étant activées simultanément à la date 0, ce temps de réponse r_2 devrait être le temps de réponse pire cas de la tâche τ_2. Autrement dit, quelque soit le scénario d'activation des tâches de cet ensemble, la tâche τ_2 ne devrait jamais obtenir un temps de réponse supérieur à celui déterminé au paragrapahe 4.b.i.2 qui vaut :

$r_2 = 8 \times C_{tick} + 7 \times C_{act} + 2 \times C_{sched} + 5 \times C_{term} + 3 \times C_{get} + 3 \times C_{rel} + 2 \times C_4 + 2 \times C_3 + C_2$

La figure 4.4 montre l'ordonnancement du même ensemble de tâches en considérant que les tâches τ_4 et τ_3 voient respectivement leur première activation apparaître aux dates $t_4 = T_{tick}$ et $t_3 = 2 \times T_{tick}$.

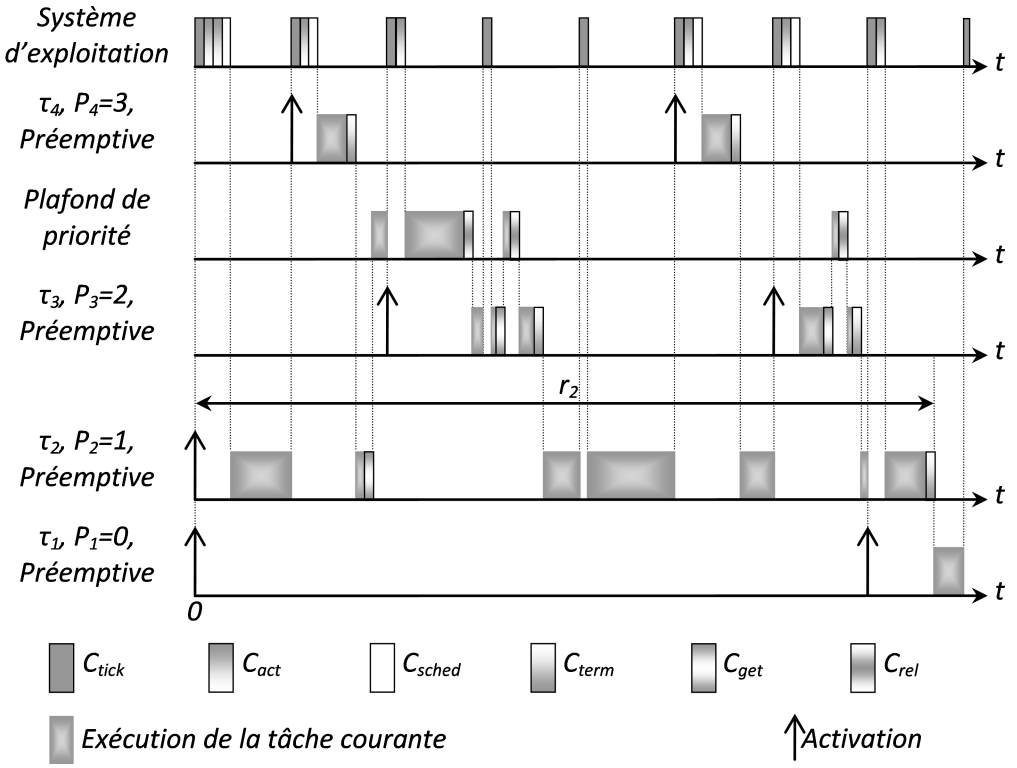

FIGURE 4.4 – Calcul d'un temps de réponse avec activations échelonnées

Au niveau du système d'exploitation, la figure 4.4 fait apparaître quatre charges C_{sched} alors que nous n'en comptions de deux à la figure 4.3. En effet, les nouvelles dates de première activation des tâches τ_4 et τ_3 occasionnent des préemptions supplémentaires. Notez qu'à sa première activation, la tâche τ_3 ne peut préempter la tâche τ_2 qui, étant alors en train d'utiliser la ressource, se trouve protégée par le mécanisme du plafond de priorité. Finalement, ce nouveau scénario mène à un temps de réponse de la tâche τ_2 supérieur au précédent de deux charges C_{sched}. L'activation simultanée de toutes les tâches n'a donc pas abouti au temps de réponse pire cas de la tâche τ_2.

Comme nous venons de le voir, la prise en compte de la charge C_{sched} dans le calcul du temps de réponse pire cas de la tâche τ_2 pose problème. En effet, si nous nous en tenons au comportement réel de la charge C_{sched}, spécifié à la sous section 3.e.i, l'intégration de cette charge nécessite la détermination exacte du nombre de préemptions. Nous retrouvons alors le problème étudié par P.Meumeu Yomsi et Y.Sorel dans leurs travaux exposés à la sous section 4.a.i. Malheureusement, l'algorithme proposé par les deux chercheurs ne s'applique qu'à des ensembles de tâches concrètes. Ce problème s'avère aujourd'hui, à notre connaissance, non résolu dans le cas de tâches non concrètes.

Revenons à l'ensemble de tâches τ qui nous occupe. Pour obtenir une borne supérieure du temps de réponse de la tâche τ_2, nous allons simplement ne pas suivre le comportement réel de la charge C_{sched} et dire que l'exécution d'une tâche est toujours précédée d'une charge C_{sched}. La figure 4.5 illustre l'ordonnancement de l'ensemble τ dans ce cas. Notez que nous avons appliqué le scénario pire cas classique en activant simultanément toutes les tâches.

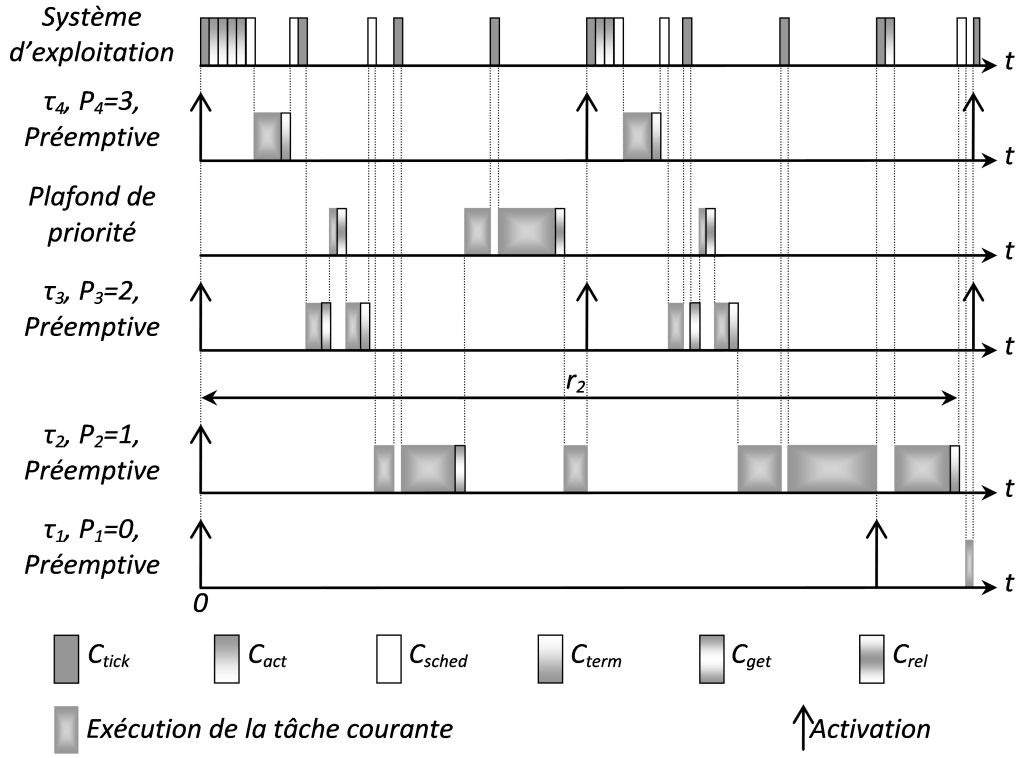

FIGURE 4.5 – Intégration de la charge C_{sched} au calcul d'un temps de réponse pire cas

Dans le cas présent, l'activation simultanée des tâches maximise leur charge de travail sur l'intervalle $[0, r_2[$. Qui plus est, leur temps d'exécution est également maximisé par l'ajout systématique d'une charge C_{sched}. Ainsi, nous obtenons bien une borne supérieure du temps de réponse de la tâche τ_2 qui, bien qu'étant légèrement pessimiste, tient parfaitement le rôle de temps de réponse pire cas. Finalement, le temps de réponse pire cas de la tâche τ_2 apparaît, à la figure 4.5, égal à :

$$r_2 = 8 \times C_{tick} + 7 \times C_{act} + 5 \times C_{sched} + 5 \times C_{term} + 3 \times C_{get} + 3 \times C_{rel} + 2 \times C_4 + 2 \times C_3 + C_2$$

4.b.iii Prise en compte des inversions de priorité

Supposons que nous étudions le temps de réponse pire cas d'une tâche τ_i appartenant à un ensemble de n tâches noté τ. Nous devons tenir compte des éventuelles inversions de priorités susceptibles de retarder le début de son exécution. Pour rappel, nous parlons d'inversion de priorité lorsqu'une tâche initialement moins prioritaire que la tâche τ_i se retrouve temporairement plus prioritaire qu'elle. Ce phénomène peut survenir lorsqu'une tâche, moins prioritaire que τ_i, est non préemptive ou utilise une resource dont la priorité de plafond est supérieure ou égale à la priorité de τ_i.

Comme expliqué à la sous section 2.a.iii, l'effet non préemptif correspond au fait qu'une tâche non préemptive puisse retarder l'exécution d'une tâche plus prioritaire. Notons $lp^{NP}(i)$ l'ensemble des tâches appartenant à τ qui soient moins prioritaires que la tâche τ_i et non préemptives. Si cet ensemble $lp^{NP}(i)$ n'est pas vide, nous devons considérer cet effet non préemptif. Pour ce faire, nous recherchons, dans cet ensemble, la tâche τ_j possédant la durée d'exécution pire cas la plus longue. Nous rappelons que, selon la sous section 3.e.i, l'exécution d'une tâche est toujours précédée d'une charge C_{term} ou C_{sched} selon que celle-ci récupère après le processeur suite à la terminaison d'une autre tâche ou à son activation par son alarme. Aussi, une tâche non préemptive ne peut monopoliser le processeur qu'à partir du moment où la charge C_{term} ou C_{sched} qui la précède a entamé son exécution. La tâche τ_j occasionne donc un retard maximum sur la tâche τ_i lorsque la charge C_{term} ou C_{sched} qui la précède entame son exécution un cycle avant l'activation de la tâche τ_i. Cependant, lors du calcul du temps de réponse pire cas de la tâche τ_i, nous ignorons si l'exécution de la tâche τ_j fait suite à une charge C_{term} ou C_{sched}. Afin de respecter notre approche pire cas, nous choisissons, parmi ces deux charges, la plus importante. Bien sûr, nous n'oublions pas que l'exécution de la tâche τ_j se termine obligatoirement une charge C_{term}. Ainsi, le retard maximal, occasionné par effet non préemptif, sur la tâche τ_i vaut :

$$\max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)$$

Sachant que nous nous intéressons à des ensembles de tâches se partageant, au plus, une ressource, comme expliqué à la sous section 3.c.ii, l'exécution de la tâche τ_i peut également être retardée par une tâche préemptive moins prioritaire qui accède à cette ressource. Bien sûr, ceci ne peut arriver que si la priorité de plafond associée à cette ressource est supérieure ou égale à la priorité de la tâche τ_i. Nous créons un ensemble, noté $lp^P_{pc}(i)$, qui est vide si la priorité de plafond de la ressource est inférieure à la priorité de la tâche τ_i. Autrement, cet ensemble regroupe les tâches préemptives appartenant à τ, moins prioritaires que la tâche τ_i, accédant à la ressource. Ainsi, si l'ensemble $lp^P_{pc}(i)$ n'est pas vide, n'importe laquelle de ses tâches est susceptible de retarder l'exécution de la tâche τ_i. Afin de déterminer le retard maximum possible, nous recherchons dans cet ensemble la tâche τ_j qui utilise la ressource le plus longtemps. La priorité de la tâche n'étant modifiée qu'à partir du moment où celle-ci a fait appel au service *GetResource*, nous supposons, afin d'obtenir le pire retard possible, que cet appel a lieu un cycle avant que la tâche τ_i ne soit activée. Une fois l'utilisation de la ressource terminée, la tâche

la libère grâce au service *ReleaseResource*. Ainsi, le retard maximal, occasionné par le mécanisme du plafond de priorité, sur la tâche τ_i vaut :

$$\max_{\tau_j \in lp^P_{pc}(i)} (C_{get} + B_j + C_{rel} - 1)$$

Deux tâches ne pouvant s'exécuter simultanément, seul un des deux retards exposés ci-dessus doit être considéré. A la section 4.c, lors du calcul du temps de réponse pire cas de la tâche τ_i, nous conserverons donc le retard le plus important :

$$\max(\max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp^P_{pc}(i)} (C_{get} + B_j + C_{rel} - 1))$$

4.b.iv Règles pour le calcul du temps de réponse pire cas d'une tâche ordonnancée par un noyau OSEK

Nous récapitulons maintenant l'ensemble des règles à respecter durant le calcul du temps de réponse pire cas de chaque tâche appartenant à un ensemble de tâches indépendantes noté τ. Nous rappelons que cette étude considère le cas où toute tâche τ_i de l'ensemble τ est périodique, préemptive ou non, non concrète et possède une échéance arbitraire. De plus, les tâches de l'ensemble τ se partagent, au plus, une ressource.

Sachant que l'ordonnancement FP/FIFO appartient aux algorithmes hybrides vus à la sous section 2.b.iii, son scénario pire cas correspond à celui exposé au paragraphe 2.c.ii.2. Dans le cas particulier d'un ordonnancement FP/FIFO, lors du calcul du temps de réponse pire cas d'une tâche τ_i, ce scénario n'impose l'activation simultanée, à la date 0, que des tâches autres que τ_i et ayant chacune une priorité fixe supérieure ou égale à celle de la tâche τ_i. La première activation de la tâche τ_i survient, quant à elle, à une date a telle que $a \in [0, T_i[$. Or, comme nous l'avons vu au paragraphe 4.b.i.2, durant le calcul du temps de réponse d'une tâche, à chacune de leurs activations, les tâches moins prioritaires apportent une charge C_{act}. Afin de maximiser leur charge et d'obtenir un scénario réellement pire cas, lors du calcul du temps de réponse pire cas de la tâche τ_i, les tâches moins prioritaires qu'elle doivent également être activées à la date 0.

Le scénario pire cas décrit au paragraphe 2.c.ii.2 impose, pour une tâche τ_i, de tester chacune des dates d'activation appartenant à l'ensemble A_i afin de trouver le temps de réponse pire cas de cette tâche. Nous rappelons que l'ensemble A_i est défini comme suit :

$$A_i = \bigcup_{a \in [0, T_i[} A_i(a) \text{ avec } A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$$

Le terme $L_i(a)$ correspond à la plus longue période d'activité de niveau de priorité $PG_i(t_i)$ obtenue lorsque la première activation de la tâche τ_i apparaît à la date a et que toutes les tâches potentiellement plus prioritaires que la tâche τ_i activée à la date t_i sont activées simultanément à la date 0. Or, dans le cas d'un ordonnancement FP/FIFO, le calcul de $L_i(a)$ revient à considérer que toute tâche, autre que τ_i, possédant une priorité fixe

supérieure ou égale à celle de τ_i est activée à la date 0. Si la tâche τ_i est activée à la date 0, nous obtenons la plus longue période d'activité de niveau de priorité P_i. Autrement dit, nous avons $L_i(0) = L_i$. Sachant que la charge liée à l'exécution de la tâche τ_i est maximisée lorsque celle-ci est activée à la date 0, nous avons, $\forall a \in [0, T_i[$, $L_i(a) \leq L_i(0) = L_i$. Ainsi, nous pouvons, sans perdre de dates d'activation, redéfinir le sous ensemble $A_i(a)$ comme suit :

$$A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i - C_i[\}$$

Cette simplification nous évite de recalculer une nouvelle période d'activité pour chaque sous ensemble $A_i(a)$.

Finalement, lors du calcul du temps de réponse pire cas d'une tâche τ_i, ordonnancée par un noyau OSEK, nous suivrons le scénario pire cas suivant :

Le temps de réponse pire cas d'une tâche τ_i, appartenant à τ, est obtenu lorsque toutes les autres tâches sont activées simultanément à la date 0. Ce temps de réponse pire cas correspond alors au temps de réponse maximum obtenu en testant toutes les dates d'activation de l'ensemble A_i défini comme suit :

$$A_i = \bigcup_{a \in [0, T_i[} A_i(a) \text{ avec } A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i - C_i[\}$$

Nous récapitulons maintenant l'ensemble des règles à suivre durant le calcul du temps de réponse pire cas de la tâche τ_i pour intégrer les charges dues au système d'exploitation :

- La charge C_{tick} doit être vue comme une tâche strictement plus prioritaire que n'importe quelle tâche de l'ensemble τ et exécutée périodiquement, avec une période égale à T_{tick}, que la tâche courante soit préemptive ou non.

- Toute tâche τ_j de l'ensemble τ voit sa durée d'exécution pire cas passer de C_j à $C_{sched} + C_j + C_{term}$. Qui plus est, dans le cas où cette tâche τ_j accèderait à la ressource, les charges C_{get} et C_{rel} seraient également ajoutées à sa durée d'exécution pire cas.

- Toute activation d'une tâche de l'ensemble τ apporte une charge C_{act}.

- Nous devons considérer le retard maximum dû à l'effet non préemptif ou au mécanisme du plafond de priorité :
  $\max(\max_{\tau_j \in lp^{NP}(i)}(\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp^{P}_{pc}(i)}(C_{get} + B_j + C_{rel} - 1))$

4.c Conditions de faisabilité réelles

A la sous section 4.c.i, nous commençons par rappeler les notations classiques en analyse temps réel. Le facteur d'utilisation et la plus longue période d'activité de niveau de priorité P_i sont respectivement étudiés aux sous sections 4.c.ii et 4.c.iii. Les sous sections 4.c.iv et 4.c.v détaillent les équations menant aux temps de réponse pires cas pour les tâches préemptives et non préemptives. Les équations proposées demeurent valables pour tout ensemble de tâches à échéance arbitraire, périodiques, indépendantes, non concrètes et ordonnancées selon la politique FP/FIFO en contexte préemptif, non préemptif ou mixte. De plus, nous ne considérons que des ensembles de tâches se partageant, au plus, une ressource.

4.c.i Notations

Nous considérons un ensemble de n tâches $\tau = \{\tau_1, ..., \tau_n\}$, soumis à un ordonnancement FP/FIFO, et une ressource protégée par la méthode du plafond de priorité exposée à la sous section 3.c.ii. Les tâches de τ qui accédent à la ressource forment un sous-ensemble noté τ_{pc}. Une tâche τ_i de τ est définie par :

- C_i : Sa durée d'exécution pire cas, ou encore WCET (Worst Case Execution Time).
- B_i : La durée pendant laquelle la tâche τ_i utilise la ressource. Cette durée est comprise dans C_i.
- T_i^* : Sa période réelle : $T_i^* = (1 + \lfloor \frac{\max(T_i - T_{tick}/2, 0)}{T_{tick}} \rfloor) T_{tick}$ (voir section 3.d)
- D_i : Son échéance temporelle : la tâche τ_i, activée à la date t, doit avoir terminé son exécution au plus tard à la date $t + D_i$.
- P_i : Paramètre supérieur ou égal à 0 et proportionnel à sa priorité.

Nous définissons également les sous-ensembles suivants :

- $hp(i)$: L'ensemble des tâches dont la priorité est strictement supérieure à celle de la tâche τ_i.
- $hp_{pc}(i)$: L'ensemble des tâches dont la priorité est strictement supérieure à celle de la tâche τ_i qui accèdent à la ressource.
- $sp(i)$: L'ensemble des tâches de même priorité que la tâche τ_i. Ce sous-ensemble exclut la tâche τ_i elle-même.
- $sp_{pc}(i)$: L'ensemble des tâches de même priorité que la tâche τ_i qui accèdent à la ressource. Ce sous-ensemble exclut la tâche τ_i elle-même.
- $lp(i)$: L'ensemble des tâches dont la priorité est strictement inférieure à celle de la tâche τ_i.
- $lp^{NP}(i)$: L'ensemble des tâches non préemptives dont la priorité est strictement inférieure à celle de la tâche τ_i.
- $lp^P_{pc}(i)$: L'ensemble des tâches préemptives dont la priorité est strictement inférieure à celle de la tâche τ_i qui accèdent à la ressource. Si la priorité de la tâche τ_i est strictement supérieure à la priorité de plafond de la ressource, alors cet ensemble est vide.
- $\tau_{i,pc}$: Contient uniquement la tâche τ_i si celle-ci accède à la ressource. Autrement cet ensemble est vide.

Nous définissons la fonction $s_i(t)$ permettant de calculer la date de la première activation, dans l'intervalle $[0, T_i^*[$, d'une tâche τ_i activée à la date t : $s_i(t) = t - \lfloor \frac{t}{T_i^*} \rfloor T_i^*$

4.c.ii Facteur d'utilisation du processeur

Reprenons la formule classique du facteur d'utilisation du processeur, présentée à la sous section 2.a.v : $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Ce facteur représente le pourcentage de temps passé par le processeur à exécuter les n tâches périodiques de l'ensemble τ. Nous allons maintenant intégrer les charges dues au système d'exploitation à ce facteur. Ainsi, nous obtiendrons le pourcentage de temps total passé à l'exécution de l'application, c'est à dire à l'exécution des n tâches de l'ensemble τ mais aussi à celle du système d'exploitation.

Modifions la formule classique en considérant que chaque tâche τ_i, de l'ensemble τ, voit sa durée d'exécution pire cas passer de C_i à $C_{act} + C_{sched} + C_i + C_{term}$. Nous arrivons alors à la formule suivante :

$$\sum_{\tau_i \in \tau} \frac{C_{act}+C_{sched}+C_i+C_{term}}{T_i^*}$$

Les tâches, qui accèdent à la ressource, requièrent les services *GetResource* puis *ReleaseResource* pour, respectivement, obtenir puis libérer celle-ci. Ces deux services amènent le terme suivant :

$$\sum_{\tau_i \in \tau_{pc}} \frac{C_{get}+C_{rel}}{T_i^*}$$

Enfin, la charge C_{tick} étant exécutée périodiquement, avec une période T_{tick}, apporte le terme suivant :

$$\frac{C_{tick}}{T_{tick}}$$

Le facteur d'utilisation du processeur, dont la complexité du calcul est en $O(n)$, pour l'exécution d'un ensemble de n tâches périodiques noté τ, vaut finalement :

$$U = \sum_{\tau_i \in \tau} \frac{C_{act}+C_{sched}+C_i+C_{term}}{T_i^*} + \sum_{\tau_i \in \tau_{pc}} \frac{C_{get}+C_{rel}}{T_i^*} + \frac{C_{tick}}{T_{tick}}$$

4.c.iii Plus longue période d'activité de niveau de priorité P_i

Nous étudions maintenant la plus longue période d'activité de même niveau de priorité que la tâche τ_i appartenant à l'ensemble τ, c'est à dire P_i. Selon la définition apportée à la sous section 2.c.i, cette période d'activité, notée L_i, définit le plus long intervalle temporel $[0, L_i[$ pendant lequel seules des tâches ayant une priorité supérieure ou égale à P_i s'exécutent. Comme expliqué dans cette même sous section, le calcul classique de L_i repose sur le scénario où l'ensemble des tâches appartenant à $\{\tau_i\} \cup sp(i) \cup hp(i)$ sont activées simultanément à la date 0. De la même façon que nous les avons ajoutées à la sous section 4.b.iv, nous devons maintenant étendre ce scénario classique aux tâches appartenant à $lp(i)$ qui devront également être activées à la date 0. Finalement, toutes les tâches de l'ensemble τ seront activées simultanément à la date 0.

Par définition, lorsqu'une tâche de priorité supérieure ou égale à P_i, c'est à dire appartenant à l'ensemble $\{\tau_i\} \cup sp(i) \cup hp(i)$, est activée durant la période d'activité $[0, L_i[$ alors cette tâche est nécessairement exécutée avant la date L_i. Le terme suivant représente l'exécution de chaque tâche appartenant à $\{\tau_i\} \cup sp(i) \cup hp(i)$ activée dans l'intervalle $[0, L_i[$:

$$\sum_{\tau_j \in \{\tau_i\} \cup sp(i) \cup hp(i)} \lceil \frac{L_i}{T_j^*} \rceil (C_{act} + C_{sched} + C_j + C_{term})$$

A chaque fois qu'une tâche de priorité supérieure ou égale à P_i, activée durant la période d'activité $[0, L_i[$, accède à la ressource, nous devons comptabiliser les services *GetResource* et *ReleaseResource* :

$$\sum_{\tau_j \in \tau_{i,pc} \cup hp_{pc}(i) \cup sp_{pc}(i)} \lceil \frac{L_i}{T_j^*} \rceil (C_{get} + C_{rel})$$

Par définition, pendant la période d'activité $[0, L_i[$, seule une tâche dont la priorité est supérieure ou égale à P_i peut s'exécuter. Malgré cela, toute tâche ayant une priorité strictement inférieure à P_i, c'est à dire appartenant à $lp(i)$, est activée à chaque fois que son alarme retentit :

$$\sum_{\tau_j \in lp(i)} \lceil \frac{L_i}{T_j^*} \rceil C_{act}$$

La gestion de la base de temps du système d'exploitation nécessite l'exécution périodique, avec une période égale à T_{tick}, de la charge C_{tick}. Ainsi, sur l'intervalle $[0, L_i[$, la charge liée à cette gestion vaut :

$$\lceil \frac{L_i}{T_{tick}} \rceil C_{tick}$$

Une fois toutes les tâches appartenant à $\{\tau_i\} \cup sp(i) \cup hp(i)$ activées à la date 0, la plus prioritaire d'entre elles peut voir son exécution retardée du fait d'une inversion de priorité. Nous devons donc intégrer le retard maximum défini à la sous section 4.b.iii :

$$\max(\max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp_{pc}^P(i)} (C_{get} + B_j + C_{rel} - 1))$$

Finalement, la date L_i se résout par la détermination du plus petit point fixe de la suite ci-dessous. Nous précisons que la complexité de ce calcul est en $O(n.(U.H))$ où U est le facteur d'utilisation du processeur et H l'hyperpériode (voir sous section 4.a.i). Nous soulignons que la durée $U.H$ constitue une borne supérieure à toute période d'activité.

$$\begin{cases} L_i^0 = \sum_{\tau_j \in \{\tau_i\} \cup sp(i) \cup hp(i)} (C_{act} + C_{sched} + C_j + C_{term}) + \sum_{\tau_j \in \tau_{i,pc} \cup hp_{pc}(i) \cup sp_{pc}(i)} (C_{get} + C_{rel}) + \\ \max(\max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp_{pc}^P(i)} (C_{get} + B_j + C_{rel} - 1)) + \\ \sum_{\tau_j \in lp(i)} C_{act} + C_{tick} \\ \\ L_i^{m+1} = \sum_{\tau_j \in \{\tau_i\} \cup sp(i) \cup hp(i)} \lceil \frac{L_i^m}{T_j^*} \rceil (C_{act} + C_{sched} + C_j + C_{term}) + \\ \sum_{\tau_j \in \tau_{i,pc} \cup hp_{pc}(i) \cup sp_{pc}(i)} \lceil \frac{L_i^m}{T_j^*} \rceil (C_{get} + C_{rel}) + \sum_{\tau_j \in lp(i)} \lceil \frac{L_i^m}{T_j^*} \rceil C_{act} + \lceil \frac{L_i^m}{T_{tick}} \rceil C_{tick} + \\ \max(\max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp_{pc}^P(i)} (C_{get} + B_j + C_{rel} - 1)) \end{cases}$$

4.c.iv Temps de réponse pire cas d'une tâche préemptive

Cette sous section détaille le calcul du temps de réponse pire cas d'une tâche périodique préemptive τ_i appartenant à un ensemble de tâches périodiques τ. Le temps de réponse de la tâche τ_i activée à la date t se note $r_{i,t}$. Comme expliqué à la sous section 4.b.iv, le temps de réponse pire cas r_i de la tâche τ_i correspond à son temps de réponse maximum observé sur l'ensemble des dates d'activation appartenant à A_i :

$$r_i = \max_{t \in A_i}(r_{i,t})$$
$$\text{Avec } A_i = \bigcup_{a \in [0, T_i^*[} A_i(a) \text{ et } A_i(a) = \{a_i^k = a + k \times T_i^*, k \in N / a_i^k \in [0, L_i - C_i[\}$$

Au paragraphe 4.c.iv.1, nous expliquons le calcul du temps de réponse de la tâche préemptive τ_i activée à la date t. Lors de ce calcul, nous prenons soin de respecter le scénario pire cas et les règles énoncées à la sous section 4.b.iv. Ainsi, alors que la première activation de la tâche τ_i intervient à la date $a \in [0, T_i^*[$, nous considérons que les autres tâches sont toujours activées simultanément à la date 0. Une fois le temps de réponse pire cas de la tâche τ_i calculé, nous présentons, au paragraphe 4.c.iv.2, les équations nécessaires à la détermination du temps passé à exécuter chaque charge du noyau durant ce temps de réponse pire cas.

4.c.iv.1 Calcul du temps de réponse de la tâche τ_i activée à la date t

Nous considérons donc une tâche préemptive τ_i activée, durant la plus longue période d'activité de niveau de priorité P_i notée L_i, à la date $t \in [0, L_i - C_i[$. Le calcul du temps de réponse $r_{i,t}$ nécessite le calcul de la date de fin d'exécution $\omega_{i,t}$. Nous commençons donc par calculer la date $\omega_{i,t}$ en suivant les règles énoncées à la sous section 4.b.iv.

Sachant que la tâche τ_i, activée à la date t, n'aura pas terminé son exécution avant la date $\omega_{i,t}$, l'ensemble des tâches plus prioritaires que τ_i activées avant cette même date seront obligatoirement exécutées dans l'intervalle $[0, \omega_{i,t}[$. Le terme suivant correspond aux exécutions des tâches strictement plus prioritaires que τ_i sur l'intervalle $[0, \omega_{i,t}[$:

$$\sum_{\tau_j \in hp(i)} \lceil \frac{\omega_{i,t}}{T_j^*} \rceil (C_{act} + C_{sched} + C_j + C_{term})$$

Dans le cas où certaines tâches strictement plus prioritaires que τ_i accèdent à la ressource, nous devons comptabiliser le coût des services *GetResource* et *ReleaseResource* :

$$\sum_{\tau_j \in hp_{pc}(i)} \lceil \frac{\omega_{i,t}}{T_j^*} \rceil (C_{get} + C_{rel})$$

Etant donné que l'ordonnancement des tâches de même priorité suit la politique FIFO, parmi les tâches de priorité P_i, seules celles activées dans l'intervalle $[0, t]$ seront exécutées et terminées avant la date $\omega_{i,t}$. Nous considérons donc que les tâches de priorité P_i activées simultanément avec τ_i, à la date t, s'exécutent avant cette dernière. De plus,

pour les tâches autres que τ_i, nous ne comptabilisons que leurs activations arrivées dans l'intervalle $[0, \omega_{i,t}[$. Concernant la tâche τ_i, nous comptabilisons ses activations sur l'intervalle $[s_i(t), t] \cap [s_i(t), \omega_{i,t}[$. Notez que la prise en compte des charges C_{act}, nécessaires aux activations des tâches de priorité P_i, n'étant pas limitée aux intervalles précédents, se fera au sein d'un terme distinct du suivant :

$$\sum_{\tau_j \in sp(i)} \min(1 + \lfloor \frac{t}{T_j^*} \rfloor, \lceil \frac{\omega_{i,t}}{T_j^*} \rceil)(C_{sched} + C_j + C_{term}) +$$
$$\min(1 + \lfloor \frac{t}{T_i^*} \rfloor, \lceil \frac{\max(\omega_{i,t} - s_i(t), 0)}{T_i^*} \rceil)(C_{sched} + C_i + C_{term})$$

De plus, lorsque des tâches de priorité P_i accèdent à la ressource, nous devons comptabiliser le coût des services *GetResource* et *ReleaseResource* :

$$\sum_{\tau_j \in sp_{pc}(i)} \min(1 + \lfloor \frac{t}{T_j^*} \rfloor, \lceil \frac{\omega_{i,t}}{T_j^*} \rceil)(C_{get} + C_{rel}) + \sigma_i \times \min(1 + \lfloor \frac{t}{T_i^*} \rfloor, \lceil \frac{\max(\omega_{i,t} - s_i(t), 0)}{T_i^*} \rceil)(C_{get} + C_{rel})$$

$$\text{où } \sigma_i = \begin{cases} 1 \text{ si } \tau_{i,pc} \neq \phi \\ 0 \text{ autrement} \end{cases}$$

Dès lors que l'alarme d'une tâche, autre que τ_i, de priorité inférieure ou égale à P_i survient dans l'intervalle $[0, \omega_{i,t}[$, indépendamment du fait que celle-ci soit ou non exécutée dans cet intervalle, son activation doit, dans tous les cas, être prise en compte. Concernant la tâche τ_i, ses activations interviennent sur l'intervalle $[s_i(t), \omega_{i,t}[$:

$$\sum_{\tau_j \in sp(i) \cup lp(i)} \lceil \frac{\omega_{i,t}}{T_j^*} \rceil C_{act} + \lceil \frac{\max(\omega_{i,t} - s_i(t), 0)}{T_i^*} \rceil C_{act}$$

Toujours sur l'intervalle $[0, \omega_{i,t}[$, l'exécution périodique de la charge C_{tick}, nécessaire à la gestion de la base de temps, apporte le terme :

$$\lceil \frac{\omega_{i,t}}{T_{tick}} \rceil C_{tick}$$

Rappelons qu'une tâche de priorité strictement inférieure à P_i peut retarder l'exécution de τ_i si cette tâche est non préemptive, ou bien si elle accède à la ressource. Le retard maximum occasionné sur la tâche τ_i vaut :

$$\max(\max_{\tau_j \in lp^{NP}(i)}(\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp_{pc}^{P}(i)}(C_{get} + B_j + C_{rel} - 1))$$

Finalement, lorsqu'une tâche préemptive τ_i est activée à la date $t \in [0, L_i - C_i[$, sa date de fin d'exécution $\omega_{i,t}$ coïncide avec le plus petit point fixe de la suite ci-dessous. Nous précisons que la complexité du calcul de $\omega_{i,t}$ est en $O(n.L)$ où L correspond à la plus longue période d'activité (voir paragraphe 2.c.i.1).

$$
\begin{cases}
\omega_{i,t}^0 = \sum\limits_{\tau_j \in hp(i)} (C_{act} + C_{sched} + C_j + C_{term}) + \sum\limits_{\tau_j \in hp_{pc}(i)} (C_{get} + C_{rel}) + \\
\sum\limits_{\tau_j \in sp(i)} (C_{act} + C_{sched} + C_j + C_{term}) + \eta_{i,t}(C_{act} + C_{sched} + C_i + C_{term}) + \\
\sum\limits_{\tau_j \in sp_{pc}(i)} (C_{get} + C_{rel}) + \eta_{i,t}\sigma_i(C_{get} + C_{rel}) + \sum\limits_{\tau_j \in lp(i)} C_{act} + C_{tick} + \\
\max(\max\limits_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max\limits_{\tau_j \in lp_{pc}^{P}(i)} (C_{get} + B_j + C_{rel} - 1)) \\
\\
\omega_{i,t}^{m+1} = \sum\limits_{\tau_j \in hp(i)} \lceil \frac{\omega_{i,t}^m}{T_j^*} \rceil (C_{act} + C_{sched} + C_j + C_{term}) + \sum\limits_{\tau_j \in hp_{pc}(i)} \lceil \frac{\omega_{i,t}^m}{T_j^*} \rceil (C_{get} + C_{rel}) + \\
\sum\limits_{\tau_j \in sp(i)} \min(1 + \lfloor \frac{t}{T_j^*} \rfloor, \lceil \frac{\omega_{i,t}^m}{T_j^*} \rceil)(C_{sched} + C_j + C_{term}) + \\
\min(1 + \lfloor \frac{t}{T_i^*} \rfloor, \lceil \frac{\max(\omega_{i,t}^m - s_i(t), 0)}{T_i^*} \rceil)(C_{sched} + C_i + C_{term}) + \\
\sum\limits_{\tau_j \in sp_{pc}(i)} \min(1 + \lfloor \frac{t}{T_j^*} \rfloor, \lceil \frac{\omega_{i,t}^m}{T_j^*} \rceil)(C_{get} + C_{rel}) + \\
\sigma_i \times \min(1 + \lfloor \frac{t}{T_i^*} \rfloor, \lceil \frac{\max(\omega_{i,t}^m - s_i(t), 0)}{T_i^*} \rceil)(C_{get} + C_{rel}) + \\
\sum\limits_{\tau_j \in sp(i) \cup lp(i)} \lceil \frac{\omega_{i,t}^m}{T_j^*} \rceil C_{act} + \lceil \frac{\max(\omega_{i,t}^m - s_i(t), 0)}{T_i^*} \rceil C_{act} + \lceil \frac{\omega_{i,t}^m}{T_{tick}} \rceil C_{tick} + \\
\max(\max\limits_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max\limits_{\tau_j \in lp_{pc}^{P}(i)} (C_{get} + B_j + C_{rel} - 1))
\end{cases}
$$

où

$$\eta_{i,t} = \begin{cases} 1 \text{ si } s_i(t) = 0 \\ 0 \text{ autrement} \end{cases} \quad \text{et } \sigma_i = \begin{cases} 1 \text{ si } \tau_{i,pc} \neq \phi \\ 0 \text{ autrement} \end{cases}$$

Si $\omega_{i,t}$ s'avère être strictement inférieur à t, cela signifie que l'instance de la tâche τ_i activée à la date t n'appartient pas à la plus longue période d'activité de niveau de priorité P_i. Dans ce cas, par défaut, le temps de réponse de la tâche τ_i vaut $r_{i,t} = C_{act} + C_{sched} + C_i + C_{term} + \sigma_i(C_{get} + C_{rel})$. Autrement, ce temps de réponse est égal à $\omega_{i,t} - t$. Nous aboutissons donc à la formule suivante :

$$r_{i,t} = \max(C_{act} + C_{sched} + C_i + C_{term} + \sigma_i(C_{get} + C_{rel}), \omega_{i,t} - t)$$

4.c.iv.2 Temps consacré à chaque charge du noyau sur le temps de réponse pire cas de la tâche τ_i

Une fois r_i connu, nous pouvons calculer, à titre purement indicatif, la durée d'exécution de chaque charge du système d'exploitation sur ce temps de réponse pire cas. Lors de la recherche de r_i, nous prendrons soin de relever la date d'activation de la tâche τ_i, notée t_i^{wcrt}, qui mène à son temps de réponse pire cas, ainsi que la date de la fin de l'exécution associée à cette activation, notée ω_i^{wcrt}. Notez que lorsqu'une tâche non préemptive est activée à une date t, le temps de réponse correspondant est, tout comme dans le cas d'une tâche préemptive, noté $r_{i,t}$. Le calcul de ce dernier est détaillé à la sous section 4.c.v.

La durée totale d'exécution de la charge C_{act}, sur l'intervalle $[t_i^{wcrt}, t_i^{wcrt} + r_i[$, provient uniquement des activations périodiques de chaque tâche de l'ensemble τ. Pour chaque tâche τ_j de l'ensemble τ autre que τ_i, nous tenons compte du décalage entre la date d'activation t_i^{wcrt} de la tâche τ_i et la première activation de la tâche τ_j à partir de cette date. Ce décalage est noté d_j. Concernant la tâche τ_i, la date t_i^{wcrt} correspondant à une de ses activations, aucun décalage n'existe. La complexité du calcul de r_i^{act} est en $O(n)$.

$$r_i^{act} = \sum_{\tau_j \in \tau - \{\tau_i\}} \lceil \frac{\max(r_i - d_j, 0)}{T_j^*} \rceil C_{act} + \lceil \frac{r_i}{T_i^*} \rceil C_{act} \text{ où } d_j = \lceil \frac{t_i^{wcrt}}{T_j^*} \rceil T_j^* - t_i^{wcrt}$$

La durée totale d'exécution de la charge C_{tick}, sur le même intervalle, découle directement de ses exécutions périodiques de période T_{tick}. Là encore, nous tenons compte du décalage, noté d_{tick}, entre la date d'activation t_i^{wcrt} de la tâche τ_i et la première exécution de la charge C_{tick} à partir de cette date. La complexité du calcul de r_i^{tick} est en $O(1)$.

$$r_i^{tick} = \lceil \frac{\max(r_i - d_{tick}, 0)}{T_{tick}} \rceil C_{tick} \text{ où } d_{tick} = \lceil \frac{t_i^{wcrt}}{T_{tick}} \rceil T_{tick} - t_i^{wcrt}$$

Rappelons que, lors du calcul de ω_i^{wcrt}, nous intégrons une charge C_{term} dès lors qu'une tâche appartenant à $hp(i)$ est activée sur l'intervalle $[0, \omega_i^{wcrt}[$. Nous comptabilisons ensuite une charge C_{term} pour chaque tâche de $sp(i)$ activée sur l'intervalle $[0, t_i^{wcrt}] \cap [0, \omega_i^{wcrt}[$. Sachant que $\omega_i^{wcrt} > t_i^{wcrt}$, l'intervalle précédent équivaut à $[0, t_i^{wcrt}]$. Concernant la tâche τ_i, nous considérons une charge C_{term} pour chacune de ses activations sur l'intervalle $[s_i(t_i^{wcrt}), t_i^{wcrt}] \cap [s_i(t_i^{wcrt}), \omega_i^{wcrt}[$ qui correspond à $[s_i(t_i^{wcrt}), t_i^{wcrt}]$. Pour chacune des activations précédentes, nous devons déterminer si la charge C_{term} s'exécute dans l'intervalle $[t_i^{wcrt}, w_i^{wcrt}[$ et si cette exécution est totale ou partielle. Supposons qu'une tâche τ_j, préemptive ou non, soit activée à la date t, nous calculons alors le temps de réponse $r_{j,t}$. Trois situations peuvent se présenter :

- La charge C_{term} qui clôture l'exécution de la tâche τ_j intervient intégralement à partir de la date t_i^{wcrt}. Autrement dit, nous avons $t_i^{wcrt} + C_{term} \leq r_{j,t} + t$. La durée r_i^{term} doit alors être incrémentée de C_{term}.
- La charge C_{term} qui clôture l'exécution de la tâche τ_j débute avant la date t_i^{wcrt} et s'achève après celle-ci. Autrement dit, nous avons $t_i^{wcrt} \leq r_{j,t} + t < t_i^{wcrt} + C_{term}$. La durée r_i^{term} doit alors être incrémentée de $r_{j,t} + t - t_i^{wcrt}$.
- La charge C_{term} qui clôture l'exécution de la tâche τ_j s'achève avant la date t_i^{wcrt}. Autrement dit, nous avons $r_{j,t} + t < t_i^{wcrt}$. La durée r_i^{term} demeure alors inchangée.

Supposons que l'exécution de la tâche τ_i soit retardée par un effet non préemptif. Nous devons alors considérer la charge C_{term} qui termine l'exécution de la tâche moins prioritaire non préemptive responsable de ce retard et, si $C_{term} \geq C_{sched}$, celle qui précède son exécution. La complexité du calcul de r_i^{term} est en $O(n^2.L)$.

$$\begin{aligned} r_i^{term} = & \sum_{\tau_j \in hp(i)} \sum_{t \in [0, \omega_i^{wcrt}[} \max(\min(r_{j,t} + t - t_i^{wcrt}, C_{term}), 0) + \\ & \sum_{\tau_j \in sp(i)} \sum_{t \in [0, t_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt}, C_{term}), 0) + \\ & \sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \max(\min(r_{i,t} + t - t_i^{wcrt}, C_{term}), 0) + \\ & \alpha_i(\max(\min(t_i^{NP} - t_i^{wcrt}, C_{term}), 0) + \beta \max(C_{term} - 1 - t_i^{wcrt}, 0)) \end{aligned}$$

$$\text{où } t_i^{NP} = \max^{NP}(i),\ \alpha_i = \begin{cases} 1 \text{ si } \max^{NP}(i) \geq \max_{pc}^{P}(i) \text{ et } lp^{NP}(i) \neq \phi \\ 0 \text{ autrement} \end{cases},$$

$$\beta = \begin{cases} 1 \text{ si } C_{term} \geq C_{sched} \\ 0 \text{ autrement} \end{cases}, \max_{pc}^{P}(i) = \max_{\tau_j \in lp_{pc}^{P}(i)} (C_{get} + B_j + C_{rel} - 1) \text{ et}$$

$$\max^{NP}(i) = \max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)$$

Chacune des activations précédentes s'accompagne également d'une charge C_{sched} qui apparaît avant même que la tâche ait entamé l'exécution de son code. Pour déterminer la charge de travail imputable à la charge C_{sched}, sur l'intervalle $[t_i^{wcrt}, w_i^{wcrt}[$, nous suivons donc le même raisonnement que précédemment. Dans le cas où l'exécution de la tâche τ_i ait été retardée par un effet non préemptif, si $C_{sched} \geq C_{term}$, nous considérons la charge C_{sched} qui précède l'exécution de la tâche moins prioritaire non préemptive responsable de ce retard. La complexité du calcul de r_i^{sched} est en $O(n^2.L)$.

$$r_i^{sched} = \sum_{\tau_j \in hp(i)} \sum_{t \in [0, \omega_i^{wcrt}[} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term} - \chi_j(C_{get} + C_{rel}) - C_j, C_{sched}), 0) +$$
$$\sum_{\tau_j \in sp(i)} \sum_{t \in [0, t_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term} - \chi_j(C_{get} + C_{rel}) - C_j, C_{sched}), 0) +$$
$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \max(\min(r_{i,t} + t - t_i^{wcrt} - C_{term} - \chi_i(C_{get} + C_{rel}) - C_i, C_{sched}), 0) +$$
$$\alpha_i \gamma \max(C_{sched} - 1 - t_i^{wcrt}, 0)$$

$$\text{où } \alpha_i = \begin{cases} 1 \text{ si } \max^{NP}(i) \geq \max_{pc}^{P}(i) \text{ et } lp^{NP}(i) \neq \phi \\ 0 \text{ autrement} \end{cases}, \gamma = \begin{cases} 1 \text{ si } C_{sched} \geq C_{term} \\ 0 \text{ autrement} \end{cases},$$

$$\chi_k = \begin{cases} 1 \text{ si } \tau_k \in \tau_{pc} \\ 0 \text{ autrement} \end{cases}, \max_{pc}^{P}(i) = \max_{\tau_j \in lp_{pc}^{P}(i)} (C_{get} + B_j + C_{rel} - 1) \text{ et}$$

$$\max^{NP}(i) = \max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)$$

Au cours de leurs exécutions, certaines des tâches précédentes peuvent accéder à la ressource. Nous commençons par comptabiliser les charges C_{rel} qui surviennent durant l'exécution de la tâche τ_i activée à la date t_i^{wcrt}. Sachant que la charge C_{rel} intervient obligatoirement avant que la tâche τ_j, activée à la date t, ne se termine, nous procédons de la même manière que précédemment, en prenant soin de soustraire la durée C_{term} à la date de fin d'exécution $r_{j,t} + t$. Supposons que l'exécution de la tâche τ_i ait été retardée par une tâche moins prioritaire préemptive protégée par le mécanisme du plafond de priorité. Dans ce cas, nous comptabilisons la charge C_{rel} qui clôture l'utilisation de la ressource par cette tâche moins prioritaire. La complexité du calcul de r_i^{rel} est en $O(n^2.L)$.

$$r_i^{rel} = \sum_{\tau_j \in hp_{pc}(i)} \sum_{t \in [0, \omega_i^{wcrt}[} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term}, C_{rel}), 0) +$$
$$\sum_{\tau_j \in sp_{pc}(i)} \sum_{t \in [0, t_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term}, C_{rel}), 0) +$$
$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \sigma_i \max(\min(r_{i,t} + t - t_i^{wcrt} - C_{term}, C_{rel}), 0) + \eta_i \max(\min(t_{pc,i}^{P} - t_i^{wcrt}, C_{rel}), 0)$$

$$\text{où } t_{pc,i}^{P} = \max_{pc}^{P}(i),\ \sigma_i = \begin{cases} 1 \text{ si } \tau_{i,pc} \neq \phi \\ 0 \text{ autrement} \end{cases}, \eta_i = \begin{cases} 1 \text{ si } \max_{pc}^{P}(i) \geq \max^{NP}(i) \text{ et } lp_{pc}^{P}(i) \neq \phi \\ 0 \text{ autrement} \end{cases},$$

$$\max_{pc}^{P}(i) = \max_{\tau_j \in lp_{pc}^{P}(i)} (C_{get} + B_j + C_{rel} - 1) \text{ et}$$

$$\max^{NP}(i) = \max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)$$

Tout appel au service *ReleaseResource* étant toujours précédé d'un appel à *GetResource*, nous nous intéressons maintenant aux charges C_{get}. N'oublions pas que la charge C_{get} intervient toujours avant que la tâche τ_j, activée à la date t, n'utilise la ressource et n'appelle les services *ReleaseResource* puis *TerminateTask*. Supposons que la tâche τ_i ait été retardée par une tâche moins prioritaire préemptive protégée par le mécanisme du plafond de priorité, nous considérons alors la charge C_{get} qui précède l'utilisation de la ressource par cette tâche. La complexité du calcul de r_i^{get} est en $O(n^2.L)$.

$$r_i^{get} = \sum_{\tau_j \in hp_{pc}(i)} \sum_{t \in [0, \omega_i^{wcrt}[} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term} - C_{rel} - B_j, C_{get}), 0) + \sum_{\tau_j \in sp_{pc}(i)} \sum_{t \in [0, t_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term} - C_{rel} - B_j, C_{get}), 0) + \sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \sigma_i \max(\min(r_{i,t} + t - t_i^{wcrt} - C_{term} - C_{rel} - B_i, C_{get}), 0) + \eta_i \max(C_{get} - 1 - t_i^{wcrt}, 0)$$

$$\text{où } \sigma_i = \begin{cases} 1 \text{ si } \tau_{i,pc} \neq \phi \\ 0 \text{ autrement} \end{cases}, \eta_i = \begin{cases} 1 \text{ si } \max_{pc}^P(i) \geq \max^{NP}(i) \text{ et } lp_{pc}^P(i) \neq \phi \\ 0 \text{ autrement} \end{cases},$$

$$\max_{pc}^P(i) = \max_{\tau_j \in lp_{pc}^P(i)} (C_{get} + B_j + C_{rel} - 1) \text{ et}$$

$$\max^{NP}(i) = \max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)$$

4.c.v Temps de réponse pire cas d'une tâche non préemptive

Cette sous section détaille le calcul du temps de réponse pire cas d'une tâche périodique non préemptive τ_i appartenant à un ensemble de tâches périodiques τ. Le temps de réponse de la tâche τ_i activée à la date t se note $r_{i,t}$. Comme expliqué à la sous section 4.b.iv, le temps de réponse pire cas r_i de la tâche τ_i correspond à son temps de réponse maximum observé sur l'ensemble des dates d'activation appartenant à A_i :

$$r_i = \max_{t \in A_i}(r_{i,t})$$

$$\text{Avec } A_i = \bigcup_{a \in [0, T_i^*[} A_i(a) \text{ et } A_i(a) = \{a_i^k = a + k \times T_i^*, k \in N / a_i^k \in [0, L_i - C_i[\}$$

Au paragraphe 4.c.v.1, nous expliquons le calcul du temps de réponse de la tâche non préemptive τ_i activée à la date t. Lors de ce calcul, nous prenons soin de respecter le scénario pire cas et les règles énoncées à la sous section 4.b.iv. Ainsi, alors que la première activation de la tâche τ_i intervient à la date $a \in [0, T_i^*[$, les autres tâches sont toujours activées simultanément à la date 0. Une fois le temps de réponse pire cas de la tâche τ_i calculé, nous présentons, au paragraphe 4.c.v.2, les équations nécessaires à la détermination du temps passé à exécuter chaque charge du noyau durant ce temps de réponse pire cas.

4.c.v.1 Calcul du temps de réponse de la tâche τ_i activée à la date t

Nous considérons donc une tâche non préemptive τ_i activée, durant la plus longue période d'activité de niveau de priorité P_i notée L_i, à la date $t \in [0, L_i - C_i[$. Dès son exécution entamée, la tâche τ_i ne peut plus être interrompue même par une tâche de plus forte priorité. Cependant, entre son activation à la date t et le début de son exécution à la date $\overline{\omega}_{i,t}$, les tâches plus prioritaires exploitent le processeur. Nous devons donc calculer la date de début d'exécution de la tâche τ_i, notée $\overline{\omega}_{i,t}$, puis la durée réelle de l'exécution qui s'en suit, notée $z_{i,t}$.

Intéressons nous, tout d'abord, à la date de début d'exécution de la tâche τ_i activée à la date $t \in [0, L_i - C_i[$. Sachant que la tâche τ_i n'entame son exécution qu'après la date $\overline{\omega}_{i,t}$, dans l'intervalle $[0, \overline{\omega}_{i,t}]$, les tâches de priorité strictement supérieure à P_i sont normalement exécutées :

$$\sum_{\tau_j \in hp(i)} (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_j^*} \rfloor)(C_{act} + C_{sched} + C_j + C_{term})$$

Dans ce même intervalle, à chaque fois qu'une tâche de $hp(i)$ accède à la ressource, nous devons comptabiliser le coût des services *GetResource* et *ReleaseResource* :

$$\sum_{\tau_j \in hp_{pc}(i)} (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_j^*} \rfloor)(C_{get} + C_{rel})$$

Toute tâche appartenant à $sp(i)$, activée dans l'intervalle $[0, t[$, est obligatoirement exécutée avant que la tâche τ_i, activée à la date t, ne débute son exécution. Dans le cas où, une tâche appartenant à $sp(i)$ serait également activée à la date t, celle-ci serait alors susceptible d'obtenir le processeur avant la tâche τ_i. De plus, nous ne considérons que les activations intervenant dans l'intervalle $[0, \overline{\omega}_{i,t}]$. Ainsi, nous considérons que toute tâche appartenant à $sp(i)$, activée dans l'intervalle $[0, t] \cap [0, \overline{\omega}_{i,t}]$, sera exécutée avant que la tâche τ_i, activée à la date t, n'obtienne le processeur. Notez que la prise en compte des charges C_{act}, nécessaires aux activations des tâches de $sp(i)$, n'étant pas limitée à l'intervalle précédent, se fera au sein d'un terme distinct du suivant :

$$\sum_{\tau_j \in sp(i)} (1 + \lfloor \frac{\min(t, \overline{\omega}_{i,t})}{T_j^*} \rfloor)(C_{sched} + C_j + C_{term})$$

Lorsqu'une tâche de $sp(i)$, activée dans l'intervalle $[0, t] \cap [0, \overline{\omega}_{i,t}]$, requière la ressource, nous devons tenir compte de ses appels aux services *GetResource* et *ReleaseResource* :

$$\sum_{\tau_j \in sp_{pc}(i)} (1 + \lfloor \frac{\min(t, \overline{\omega}_{i,t})}{T_j^*} \rfloor)(C_{get} + C_{rel})$$

Sachant que nous sommes en train de calculer la date $\overline{\omega}_{i,t}$ après laquelle la tâche τ_i, activée à la date t, débute son exécution, nous ne devons comptabiliser dans ce calcul que les exécutions des activations de τ_i survenues dans l'intervalle $[s_i(t), t[\cap [s_i(t), \overline{\omega}_{i,t}]$:

$$\min(\lfloor \frac{t}{T_i^*} \rfloor, \mu_{i,t}(\overline{\omega}_{i,t}))(C_{sched} + C_i + C_{term}) \text{ où}$$

$$\mu_{i,t}(\overline{\omega}_{i,t}) = \begin{cases} 1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T_i^*} \rfloor \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \text{ et } 0 \text{ autrement} \end{cases}$$

Bien sûr, si la tâche τ_i utilise la ressource, nous n'oublierons pas d'ajouter le coût des services *GetResource* et *ReleaseResource* pour chacune de ses activations dans $[s_i(t), t[\cap[s_i(t), \overline{\omega}_{i,t}]$:

$$\sigma_i \times \min(\lfloor \frac{t}{T_i^*} \rfloor, \mu_{i,t}(\overline{\omega}_{i,t}))(C_{get} + C_{rel}) \text{ où}$$

$$\mu_{i,t}(\overline{\omega}_{i,t}) = \begin{cases} 1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T_i^*} \rfloor \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \text{ et 0 autrement} \end{cases} \quad \text{et } \sigma_i = \begin{cases} 1 \text{ si } \tau_{i,pc} \neq \phi \\ 0 \text{ autrement} \end{cases}$$

Qu'elles soient exécutées ou non dans l'intervalle $[0, \overline{\omega}_{i,t}]$, les activations de toute tâche, de priorité inférieure ou égale à P_i, dans cet intervalle, doivent être prises en compte. Concernant la tâche τ_i, nous comptabilisons ses activations sur l'intervalle $[s_i(t), \overline{\omega}_{i,t}]$:

$$\sum_{\tau_j \in sp(i) \cup lp(i)} (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_j^*} \rfloor) C_{act} + \mu_{i,t}(\overline{\omega}_{i,t}) C_{act} \text{ où}$$

$$\mu_{i,t}(\overline{\omega}_{i,t}) = \begin{cases} 1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T_i^*} \rfloor \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \text{ et 0 autrement} \end{cases}$$

Sur ce même intervalle $[0, \overline{\omega}_{i,t}]$, l'exécution périodique de la charge C_{tick} nécessaire à la gestion de la base de temps et des alarmes amène :

$$(1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_{tick}} \rfloor) C_{tick}$$

Rappelons qu'une tâche de priorité strictement inférieure à P_i peut retarder l'exécution de τ_i si cette tâche est non préemptive, ou bien si elle accède à la ressource. Le retard maximum occasionné sur la tâche τ_i vaut :

$$\max(\max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp^P_{pc}(i)} (C_{get} + B_j + C_{rel} - 1))$$

Finalement, lorsque la tâche non préemptive τ_i est activée à la date t, son exécution ne débute qu'après la date $\overline{\omega}_{i,t}$ qui correspond au plus petit point fixe de la suite ci-dessous. La complexité du calcul de $\overline{\omega}_{i,t}$ est en $O(n.L)$.

$$\begin{cases} \overline{\omega}^0_{i,t} = \sum_{\tau_j \in hp(i)} (C_{act} + C_{sched} + C_j + C_{term}) + \sum_{\tau_j \in hp_{pc}(i)} (C_{get} + C_{rel}) + \\ \sum_{\tau_j \in sp(i)} (C_{act} + C_{sched} + C_j + C_{term}) + \sum_{\tau_j \in sp_{pc}(i)} (C_{get} + C_{rel}) + \eta_{i,t}(C_{act} + C_{sched} + C_i + C_{term}) + \\ \eta_{i,t}\sigma_i(C_{get} + C_{rel}) + \sum_{\tau_j \in lp(i)} C_{act} + C_{tick} + \\ \max(\max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp^P_{pc}(i)} (C_{get} + B_j + C_{rel} - 1)) \\ \\ \overline{\omega}^{m+1}_{i,t} = \sum_{\tau_j \in hp(i)} (1 + \lfloor \frac{\overline{\omega}^m_{i,t}}{T_j^*} \rfloor)(C_{act} + C_{sched} + C_j + C_{term}) + \sum_{\tau_j \in hp_{pc}(i)} (1 + \lfloor \frac{\overline{\omega}^m_{i,t}}{T_j^*} \rfloor)(C_{get} + C_{rel}) + \\ \sum_{\tau_j \in sp(i)} (1 + \lfloor \frac{\min(t, \overline{\omega}^m_{i,t})}{T_j^*} \rfloor)(C_{sched} + C_j + C_{term}) + \sum_{\tau_j \in sp_{pc}(i)} (1 + \lfloor \frac{\min(t, \overline{\omega}^m_{i,t})}{T_j^*} \rfloor)(C_{get} + C_{rel}) + \\ \min(\lfloor \frac{t}{T_i^*} \rfloor, \mu_{i,t}(\overline{\omega}^m_{i,t}))(C_{sched} + C_i + C_{term}) + \sigma_i \times \min(\lfloor \frac{t}{T_i^*} \rfloor, \mu_{i,t}(\overline{\omega}^m_{i,t}))(C_{get} + C_{rel}) + \\ \sum_{\tau_j \in sp(i) \cup lp(i)} (1 + \lfloor \frac{\overline{\omega}^m_{i,t}}{T_j^*} \rfloor) C_{act} + \mu_{i,t}(\overline{\omega}^m_{i,t}) C_{act} + (1 + \lfloor \frac{\overline{\omega}^m_{i,t}}{T_{tick}} \rfloor) C_{tick} + \\ \max(\max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1), \max_{\tau_j \in lp^P_{pc}(i)} (C_{get} + B_j + C_{rel} - 1)) \end{cases}$$

où $\mu_{i,t}(\overline{\omega}_{i,t}) = \begin{cases} 1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T_i^*} \rfloor \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \text{ et 0 autrement} \end{cases}$, $\sigma_i = \begin{cases} 1 \text{ si } \tau_{i,pc} \neq \phi \\ 0 \text{ autrement} \end{cases}$ et

$$\eta_{i,t} = \begin{cases} 1 \text{ si } s_i(t) = 0 \\ 0 \text{ autrement} \end{cases}$$

Maintenant que nous connaissons la date après laquelle la tâche non préemptive τ_i, activée à la date t, débute son exécution, nous nous intéressons à la durée de celle-ci notée $z_{i,t}$. L'activation de la tâche τ_i à la date t ayant déjà été prise en compte lors du calcul de la date $\overline{\omega}_{i,t}$, concernant son exécution, nous ne devons plus considérer que :

$$C_{sched} + C_i + C_{term}$$

Dans le cas où τ_i accède à la ressource, nous intégrons également le coût des services *GetResource* et *ReleaseResource* :

$$\sigma_i(C_{get} + C_{rel}) \text{ où } \sigma_i = \left\{ \begin{array}{l} 1 \text{ si } \tau_{i,pc} \neq \phi, \\ 0 \text{ autrement} \end{array} \right.$$

Durant son exécution, la tâche τ_i ne peut en aucun cas être préemptée par une autre tâche quelque soit sa priorité. Cependant, le système d'exploitation active les tâches, au fur et à mesure de leurs activations, en vue de leurs prochaines exécutions. La prise en compte de ces activations nécessite de déterminer, pour chaque tâche τ_j de l'ensemble τ, le décalage, noté t^0_j, entre la date $\overline{\omega}_{i,t}$ et sa première activation après celle-ci. La prise en compte des activations des tâches de l'ensemble τ, pendant l'exécution de la tâche τ_i, correspond alors au terme suivant :

$$\sum_{\tau_j \in \tau} \lceil \frac{\max(z_{i,t} - t^0_j, 0)}{T^*_j} \rceil C_{act} \text{ où } t^0_{j \neq i} = (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T^*_j} \rfloor) T^*_j - \overline{\omega}_{i,t} \text{ et}$$

$$t^0_i = \left\{ \begin{array}{l} (1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T^*_i} \rfloor) T^*_i - (\overline{\omega}_{i,t} - s_i(t)) \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \\ s_i(t) - \overline{\omega}_{i,t} \text{ autrement} \end{array} \right.$$

Enfin, nous intégrons le fait que, malgré l'exécution de la tâche τ_i, le système d'exploitation se doit de maintenir sa base de temps. Là encore, nous tenons compte du décalage entre la date $\overline{\omega}_{i,t}$ et la première arrivée de la charge C_{tick} après cette date.

$$\lceil \frac{\max(z_{i,t} - t^0_{tick}, 0)}{T_{tick}} \rceil C_{tick} \text{ où } t^0_{tick} = (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_{tick}} \rfloor) T_{tick} - \overline{\omega}_{i,t}$$

Ainsi, la durée de l'exécution de la tâche τ_i, activée à la date t, coïncide avec le plus petit point fixe de la suite ci-dessous. La complexité du calcul de $z_{i,t}$ est en $O(n.L)$.

$$\left\{ \begin{array}{l} z^0_{i,t} = C_{sched} + C_i + C_{term} + \sigma_i(C_{get} + C_{rel}) \\ z^{m+1}_{i,t} = C_{sched} + C_i + C_{term} + \sigma_i(C_{get} + C_{rel}) + \\ \sum_{\tau_j \in \tau} \lceil \frac{\max(z^m_{i,t} - t^0_j, 0)}{T^*_j} \rceil C_{act} + \lceil \frac{\max(z^m_{i,t} - t^0_{tick}, 0)}{T_{tick}} \rceil C_{tick} \end{array} \right.$$

$$\text{où } t^0_{j \neq i} = (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T^*_j} \rfloor) T^*_j - \overline{\omega}_{i,t}, \; t^0_i = \left\{ \begin{array}{l} (1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T^*_i} \rfloor) T^*_i - (\overline{\omega}_{i,t} - s_i(t)) \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \\ s_i(t) - \overline{\omega}_{i,t} \text{ autrement} \end{array} \right. ,$$

$$t^0_{tick} = (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_{tick}} \rfloor) T_{tick} - \overline{\omega}_{i,t} \text{ et } \sigma_i = \left\{ \begin{array}{l} 1 \text{ si } \tau_{i,pc} \neq \phi, \\ 0 \text{ autrement} \end{array} \right.$$

Si $\overline{\omega}_{i,t} + z_{i,t}$ s'avère être strictement inférieur à t, cela signifie que l'instance de la tâche τ_i activée à la date t n'appartient pas à la plus longue période d'activité de niveau de priorité P_i. Dans ce cas, par défaut, le temps de réponse de la tâche τ_i vaut $r_{i,t} = C_{act} + C_{sched} + C_i + C_{term} + \sigma_i(C_{get} + C_{rel})$. Autrement, ce temps de réponse est égal à $\overline{\omega}_{i,t} + z_{i,t} - t$. Nous aboutissons donc à la formule suivante :

$$r_{i,t} = \max(C_{act} + C_{sched} + C_i + C_{term} + \sigma_i(C_{get} + C_{rel}), \overline{\omega}_{i,t} + z_{i,t} - t) \text{où}$$
$$\sigma_i = \left\{ \begin{array}{l} 1 \text{ si } \tau_{i,pc} \neq \phi, \\ 0 \text{ autrement} \end{array} \right.$$

4.c.v.2 Temps consacré à chaque charge du noyau sur le temps de réponse pire cas de la tâche τ_i

Nous noterons t_i^{wcrt} la date d'activation qui mène au temps de réponse pire cas de la tâche τ_i. La date à laquelle la tâche τ_i débute son exécution et la durée de celle-ci, quand τ_i est activée à la date t_i^{wcrt}, seront respectivement notées $\overline{\omega}_i^{wcrt}$ et z_i^{wcrt}. Comme nous l'avons fait au paragraphe 4.c.iv.2 dans le cas d'une tâche préemptive, nous calculons, toujours à titre purement indicatif, sur le temps de réponse pire cas r_i de la tâche non préemptive τ_i, la durée d'exécution de chaque charge du système d'exploitation. Nous rappelons que le temps de réponse d'une tâche préemptive τ_i, activée à une date t, se note également $r_{i,t}$ et que son calcul est détaillé au paragraphe 4.c.iv.1.

Comme dans le cas d'une tâche préemptive, étudié au paragraphe 4.c.iv.2, la durée totale d'exécution de la charge C_{act}, sur l'intervalle $[t_i^{wcrt}, t_i^{wcrt} + r_i[$, ne provient que des activations périodiques de chaque tâche de l'ensemble τ. De même, la durée totale d'exécution de la charge C_{tick}, sur le même intervalle, découle, là encore, de ses exécutions périodiques de période T_{tick}. Pour ces deux cas, nous nous reportons donc aux formules fournies au paragraphe 4.c.iv.2.

Rappelons que, lors du calcul de r_i, nous intégrons une charge C_{term} dès lors qu'une tâche appartenant à $hp(i)$ est activée sur l'intervalle $[0, \overline{\omega}_i^{wcrt}]$. Nous comptabilisons ensuite une charge C_{term} pour chaque tâche de $sp(i)$ activée sur l'intervalle $[0, t_i^{wcrt}] \cap [0, \overline{\omega}_i^{wcrt}]$. Sachant que $\overline{\omega}_i^{wcrt} \geq t_i^{wcrt}$, l'intervalle précédent équivaut à $[0, t_i^{wcrt}]$. Concernant la tâche τ_i, nous considérons une charge C_{term} pour chacune de ses activations sur l'intervalle $[s_i(t), t_i^{wcrt}] \cap [0, \overline{\omega}_i^{wcrt}]$ qui correspond à $[s_i(t_i^{wcrt}), t_i^{wcrt}]$. Supposons que l'exécution de la tâche τ_i soit retardée par un effet non préemptif. Nous devons alors considérer la charge C_{term} qui termine l'exécution de la tâche moins prioritaire non préemptive responsable de ce retard et, si $C_{term} \geq C_{sched}$, celle qui précède son exécution. La complexité du calcul de r_i^{term} est en $O(n^2.L)$.

$$
\begin{aligned}
r_i^{term} = & \sum_{\tau_j \in hp(i)} \sum_{t \in [0, \overline{\omega}_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt}, C_{term}), 0) + \\
& \sum_{\tau_j \in sp(i)} \sum_{t \in [0, t_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt}, C_{term}), 0) + \\
& \sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \max(\min(r_{i,t} + t - t_i^{wcrt}, C_{term}), 0) + \\
& \alpha_i(\max(\min(t_i^{NP} - t_i^{wcrt}, C_{term}), 0) + \beta \max(C_{term} - 1 - t_i^{wcrt}, 0))
\end{aligned}
$$

$$
\text{où } t_i^{NP} = \max^{NP}(i),\ \alpha_i = \begin{cases} 1 \text{ si } \max^{NP}(i) \geq \max_{pc}^{P}(i) \text{ et } lp^{NP}(i) \neq \phi \\ 0 \text{ autrement} \end{cases},
$$

$$
\beta = \begin{cases} 1 \text{ si } C_{term} \geq C_{sched} \\ 0 \text{ autrement} \end{cases},\ \max_{pc}^{P}(i) = \max_{\tau_j \in lp_{pc}^{P}(i)} (C_{get} + B_j + C_{rel} - 1) \text{ et}
$$

$$
\max^{NP}(i) = \max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)
$$

Chacune des activations précédentes s'accompagne également d'une charge C_{sched} qui apparaît avant même que la tâche ait entamé l'exécution de son code. Pour déterminer

la charge de travail imputable à la charge C_{sched}, sur l'intervalle $[t_i^{wcrt}, \overline{\omega}_i^{wcrt} + z_i^{wcrt}[$, nous suivons donc le même raisonnement que précédemment. Dans le cas où l'exécution de la tâche τ_i ait été retardée par un effet non préemptif, si $C_{sched} \geq C_{term}$, nous considérons la charge C_{sched} qui précède l'exécution de la tâche moins prioritaire non préemptive responsable de ce retard. La complexité du calcul de r_i^{sched} est en $O(n^2.L)$.

$$r_i^{sched} = \sum_{\tau_j \in hp(i)} \sum_{t \in [0, \overline{\omega}_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term} - \chi_j(C_{get} + C_{rel}) - C_j, C_{sched}), 0) +$$
$$\sum_{\tau_j \in sp(i)} \sum_{t \in [0, t_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term} - \chi_j(C_{get} + C_{rel}) - C_j, C_{sched}), 0) +$$
$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \max(\min(r_{i,t} + t - t_i^{wcrt} - \chi_i(C_{get} + C_{rel}) - C_{term} - C_i, C_{sched}), 0) +$$
$$\alpha_i \gamma \max(C_{sched} - 1 - t_i^{wcrt}, 0)$$

$$\text{où } \alpha_i = \begin{cases} 1 \text{ si } \max^{NP}(i) \geq \max_{pc}^{P}(i) \text{ et } lp^{NP}(i) \neq \phi \\ 0 \text{ autrement} \end{cases}, \gamma = \begin{cases} 1 \text{ si } C_{sched} \geq C_{term} \\ 0 \text{ autrement} \end{cases},$$

$$\chi_k = \begin{cases} 1 \text{ si } \tau_k \in \tau_{pc} \\ 0 \text{ autrement} \end{cases}, \max_{pc}^{P}(i) = \max_{\tau_j \in lp_{pc}^{P}(i)} (C_{get} + B_j + C_{rel} - 1) \text{ et}$$
$$\max^{NP}(i) = \max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)$$

Au cours de leurs exécutions, certaines des tâches précédentes peuvent accéder à la ressource. Nous commençons par comptabiliser les charges C_{rel} qui surviennent durant l'exécution de la tâche τ_i activée à la date t_i^{wcrt}. Sachant que la charge C_{rel} intervient obligatoirement avant que la tâche τ_j, activée à la date t, ne se termine, nous procédons de la même manière que précédemment, en prenant soin de soustraire la durée C_{term} à la date de fin d'exécution $r_{j,t} + t$. Supposons que l'exécution de la tâche τ_i ait été retardée par une tâche moins prioritaire préemptive protégée par le mécanisme du plafond de priorité. Dans ce cas, nous comptabilisons la charge C_{rel} qui clôture l'utilisation de la ressource par cette tâche moins prioritaire. La complexité du calcul de r_i^{rel} est en $O(n^2.L)$.

$$r_i^{rel} = \sum_{\tau_j \in hp_{pc}(i)} \sum_{t \in [0, \overline{\omega}_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term}, C_{rel}), 0) +$$
$$\sum_{\tau_j \in sp_{pc}(i)} \sum_{t \in [0, t_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term}, C_{rel}), 0) +$$
$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \sigma_i \max(\min(r_{i,t} + t - t_i^{wcrt} - C_{term}, C_{rel}), 0) + \eta_i \max(\min(t_{pc,i}^{P} - t_i^{wcrt}, C_{rel}), 0)$$

$$\text{où } t_{pc,i}^{P} = \max_{pc}^{P}(i), \sigma_i = \begin{cases} 1 \text{ si } \tau_{i,pc} \neq \phi \\ 0 \text{ autrement} \end{cases}, \eta_i = \begin{cases} 1 \text{ si } \max_{pc}^{P}(i) \geq \max^{NP}(i) \text{ et } lp_{pc}^{P}(i) \neq \phi \\ 0 \text{ autrement} \end{cases},$$

$$\max_{pc}^{P}(i) = \max_{\tau_j \in lp_{pc}^{P}(i)} (C_{get} + B_j + C_{rel} - 1) \text{ et}$$
$$\max^{NP}(i) = \max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)$$

Tout appel au service *ReleaseResource* étant toujours précédé d'un appel à *GetResource*, nous nous intéressons maintenant aux charges C_{get}. N'oublions pas que la charge C_{get} intervient toujours avant que la tâche τ_j, activée à la date t, n'utilise la ressource et n'appelle les services *ReleaseResource* puis *TerminateTask*. Supposons que la tâche τ_i ait été retardée par une tâche moins prioritaire préemptive protégée par le mécanisme du plafond de priorité, nous considérons alors la charge C_{get} qui précède l'utilisation de la ressource par cette tâche. La complexité du calcul de r_i^{get} est en $O(n^2.L)$.

$$r_i^{get} = \sum_{\tau_j \in hp_{pc}(i)} \sum_{t \in [0, \overline{\omega}_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term} - C_{rel} - B_j, C_{get}), 0) +$$
$$\sum_{\tau_j \in sp_{pc}(i)} \sum_{t \in [0, t_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt} - C_{term} - C_{rel} - B_j, C_{get}), 0) +$$
$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \sigma_i \max(\min(r_{i,t} + t - t_i^{wcrt} - C_{term} - C_{rel} - B_i, C_{get}), 0) +$$
$$\eta_i \max(C_{get} - 1 - t_i^{wcrt}, 0)$$

$$\text{où } \sigma_i = \begin{cases} 1 \text{ si } \tau_{i,pc} \neq \phi \\ 0 \text{ autrement} \end{cases}, \eta_i = \begin{cases} 1 \text{ si } \max_{pc}^P(i) \geq \max^{NP}(i) \text{ et } lp_{pc}^P(i) \neq \phi \\ 0 \text{ autrement} \end{cases},$$

$$\max_{pc}^P(i) = \max_{\tau_j \in lp_{pc}^P(i)} (C_{get} + B_j + C_{rel} - 1) \text{ et}$$

$$\max^{NP}(i) = \max_{\tau_j \in lp^{NP}(i)} (\max(C_{term}, C_{sched}) + C_j + C_{term} - 1)$$

4.d Extension aux tâches sporadiques et aux interruptions

A la section précédente, nous avons étudié les équations nécessaires au calcul du temps de réponse pire cas r_i d'une tâche périodique τ_i que celle-ci soit préemptive ou non. Comme expliqué à la section 2.d, une condition nécessaire et suffisante à la faisabilité d'un ensemble de n tâches périodiques $\tau = \{\tau_1, ..., \tau_n\}$ consiste à vérifier que $\forall \tau_i \in \tau, r_i \leq D_i$ et $U \leq 1$. Nous souhaitons maintenant étendre cette étude en intégrant des tâches sporadiques ainsi que des interruptions.

Tout d'abord, nous rappelons qu'une tâche τ_i est dite sporadique si, et seulement si, deux activations successives de τ_i sont toujours espacées d'un temps supérieur ou égal à T_i. De toute évidence, l'impact d'une tâche sporadique, sur les temps de réponse des autres tâches du système, est d'autant plus fort que ses activations sont fréquentes. Ainsi, selon notre approche pire cas, prendre en compte une tâche sporadique τ_i revient simplement à la considérer comme une tâche périodique de période T_i.

Concernant les interruptions, celles-ci peuvent être périodiques, comme dans le cas du compteur qui génère la base de temps de notre exécutif OSEK, ou bien apériodiques. Dans le cas d'une interruption périodique, celle-ci peut être assimilée à une tâche de priorité strictement supérieure à celle de n'importe quelle tâche du système. Les interruptions apériodiques, quant à elles, doivent être gérées selon le modèle fenêtré : pour une interruption donnée, nous autorisons au plus k occurrences de celle-ci dans tout intervalle de temps de largeur W. Autrement dit, quelque ce soit la date t, au plus, k arrivées de l'interruption apériodique concernée peuvent advenir dans l'intervalle $[t, t + W[$. Lors du calcul du temps de réponse pire cas, d'une tâche τ_i, nous considérons, conformément au scénario pire cas décrit à la sous section 4.b.iv, que toutes les tâches de priorité supérieure à P_i sont activées simultanément à la date 0. Or, la priorité d'une interruption quelconque est strictement supérieure à celle de n'importe quelle tâche. Ainsi, nous respecterons le scénario pire cas en considérant que les k occurrences de notre interruption apériodique surviennent toujours simultanément à partir de la date 0 puis périodiquement avec une

période égale à W. Notons C la durée d'exécution pire cas de notre interruption apériodique. La prise en compte de celle-ci équivaut à l'assimiler à une tâche périodique de priorité $P = +\infty$, de période $T = W$ et dont la durée d'exécution pire cas vaut $k \times C$.

A la section 3.a, nous distinguons les interruptions de type ISR1 de celles de type ISR2. Pour mémoire, par rapport à celles de type ISR1, les interruptions de type ISR2 peuvent appeler certains services du noyau et sont vues par celui-ci. C'est pourquoi, la prise en compte des interruptions de type ISR2 requière la caractérisation de deux nouvelles charges nécessaires au noyau pour démarrer proprement ce type d'interruptions puis pour en sortir.

4.e Synthèse

Dans ce chapitre, nous avons exposé différents travaux de recherche visant à intégrer le coût du système d'exploitation dans les conditions de faisabilité. Puis, nous avons développé les équations nécessaires au calcul des temps de réponse pires cas de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes, non concrètes et se partageant, au plus, une ressource. Ces équations améliorent les outils de dimensionnement temps réel actuels en tenant compte de l'exécution du système d'exploitation. Nous expliquons également comment considérer, du fait de notre approche pire cas, des tâches sporadiques et des interruptions en les assimilant à des tâches périodiques. Au chapitre 6, nous expérimenterons les équations précédentes sur un ensemble de dix tâches périodiques avec un paramètre T_{tick} variant de $100\mu s$ à $1000\mu s$. A cette occasion, nous constaterons leur efficacité pour le dimensionnement temps réel d'applications basées sur un noyau OSEK.

Chapitre 5

Conditions de faisabilité avec l'ordonnancement EDF

Comme nous l'expliquons au chapitre 2, la politique d'ordonnancement EDF, basée sur des priorités dynamiques, s'avère plus optimale que la politique FP/FIFO prescrite par le standard OSEK et basée sur des priorités fixes. Autrement dit, tout ensemble de tâches ordonnançable avec la politique FP/FIFO l'est obligatoirement avec la politique EDF alors que l'inverse est faux. C'est pourquoi, nous avons souhaité implanter l'algorithme d'ordonnancement EDF sur notre exécutif OSEK. La section 5.a explique comment nous avons déployé cet algorithme en nous basant sur les seules tâches à priorité fixe autorisées par le standard OSEK. Notre implémentation permet l'ordonnancement de tout ensemble de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes, non concrètes et n'utilisant aucune ressource. Cette section détaille également les charges cumulées du noyau et de notre algorithme. A la section 5.b, nous précisons comment les charges précédentes s'intègrent dans un calcul de temps de réponse pire cas ainsi que le scénario pire cas à respecter. Puis, la section 5.c rappelle les notations temps réel nécessaires à l'analyse de la politique EDF. Enfin, les calculs des temps de réponse pires cas pour des tâches préemptives et non préemptives sont respectivement étudiés aux sections 5.d et 5.e.

5.a Mise en oeuvre de l'algorithme EDF

Cette section traite du déploiement de la politique d'ordonnancement EDF sur un noyau OSEK ne gérant que des tâches à priorité fixe. La sous section 5.a.i expose l'algorithme mis en oeuvre. Puis, les charges apportées par celui-ci sont détaillées à la sous section 5.a.ii.

5.a.i Déploiement sur des tâches à priorité fixe

Soit un ensemble de n tâches périodiques, ordonnancées selon l'algorithme EDF, noté $\rho = \{\rho_1, ..., \rho_n\}$. Chaque tâche ρ_i, appartenant à ρ, peut être préemptive ou non préemptive et se caractérise par :

- C_i : Sa durée d'exécution pire cas, ou encore WCET (Worst Case Execution Time).
- T_i : Sa période.
- D_i : Son échéance temporelle : la tâche τ_i, activée à la date t, doit avoir terminé son exécution au plus tard à la date $t + D_i$.
- F_i : La tâche τ_i est préemptive si ce booléen est vrai et non préemptive autrement.

L'absence de priorité parmi les paramètres de la tâche ρ_i tient au fait que l'algorithme EDF lui alloue une nouvelle priorité dynamique à chacune de ses activations. Pour rappel, lorsque la tâche ρ_i est activée à la date t_i, sa priorité dynamique est égale à son échéance absolue, c'est à dire $t_i + D_i$. Comme expliqué au paragraphe 2.b.ii.1, la politique d'ordonnancement EDF consiste à élire, parmi les tâches ayant été activées, celle possédant l'échéance absolue la plus proche, c'est à dire celle dont la priorité dynamique est la plus faible. Prenons pour exemple l'ensemble décrit au tableau 5.1 et illustré à la figure 5.1.

Tâche	*C*	*D*	*T*	*F*
ρ_1	3	7	8	true
ρ_2	1	3	3	false
ρ_3	1	6	7	true

TABLE 5.1 – Illustration de la mise en oeuvre de l'ordonnancement EDF à travers des tâches à priorité fixe : caractéristiques des tâches de l'exemple

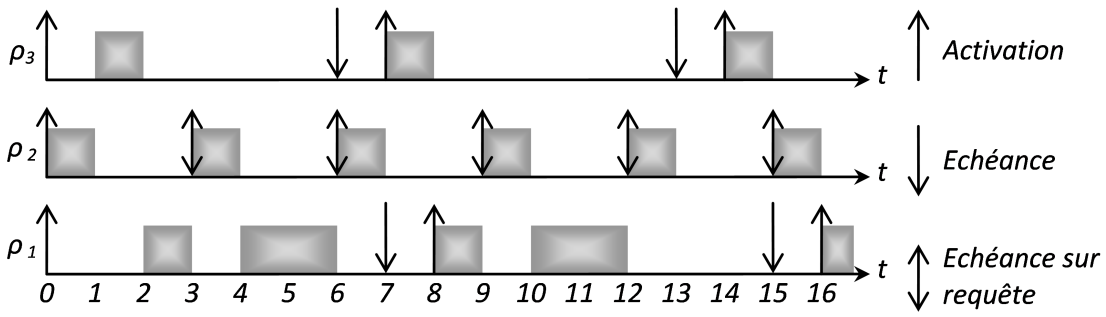

FIGURE 5.1 – Ordonnancement EDF de l'ensemble ρ décrit au tableau 5.1

Sachant que le standard OSEK ne prévoit que l'utilisation de tâches à priorité fixe ordonnancées selon FP/FIFO, notre objectif consiste à ordonnancer l'ensemble ρ avec l'algorithme EDF en exécutant ses tâches à travers des tâches à priorité fixe. Etant donné que les activations d'une même tâche s'exécutent toujours chacune leur tour, nous aurons besoin, au plus, de n tâches à priorité fixe pour exécuter les n tâches de l'ensemble ρ en respectant l'algorithme EDF. Ainsi, nous créons un ensemble de n tâches à priorité fixe $\tau = \{\tau_1, ..., \tau_n\}$ tel que $\forall \tau_i \in \tau, P_i = i$. Ainsi, toute tâche τ_i de l'ensemble τ est d'autant plus prioritaire que son indice est élevé. La figure 5.2 illustre l'ordonnancement EDF de l'ensemble ρ décrit au tableau 5.1 sur un ensemble τ de trois tâches à priorité fixe.

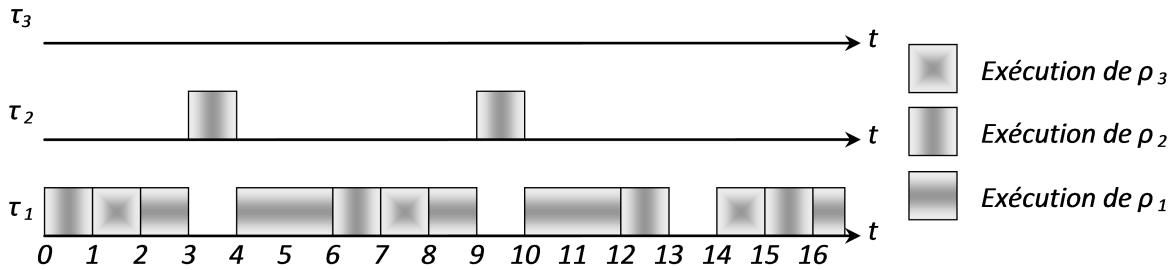

FIGURE 5.2 – Ordonnancement EDF à travers des tâches à priorité fixe

Nous précisons que notre implémentation de l'algorithme EDF conserve, pour chaque tâche ρ_i appartenant à ρ, les trois états possibles définis par le standard OSEK :

- **Courant** : C'est l'état de la tâche à laquelle le processeur a été alloué afin qu'elle s'exécute. Cette tâche correspond, sauf en présence d'un effet non préemptif, à celle possédant l'échéance absolue la plus proche. A un instant donné, une seule tâche peut occuper cet état. Au contraire, les deux états suivants peuvent être adoptés simultanément par plusieurs tâches.

- **Prêt** : La tâche a été activée et attend que l'ordonnanceur l'élise en la plaçant à l'état *courant*.

- **Suspendu** : La tâche est passive et passera à l'état *prêt* à sa prochaine activation.

Lorsqu'une tâche ρ_i, appartenant à ρ, est activée, la placer à l'état *prêt* revient simplement à l'insérer dans une file d'attente. Afin que l'algorithme EDF puisse s'accomplir correctement, lors de l'insertion d'une tâche dans cette file d'attente, nous devons préciser la priorité dynamique associée à son activation. C'est pourquoi nous utilisons une structure nommée *TachePrete* contenant trois champs : l'indice de la tâche noté *id*, la priorité dynamique de l'instance correspondante notée P ainsi qu'un booléen, noté f, indiquant si la tâche est préemptive ou non. Ainsi, la file d'attente est constituée de structures de type *TachePrete*. Enfin, à chaque fois qu'une tâche ρ_i de l'ensemble ρ est activée, nous réactualisons la variable notée t_i^{act} qui précise la date de sa prochaine activation.

Comme expliqué précédemment, l'ordonnancement selon la politique EDF d'un ensemble de n tâches périodiques ρ s'appuie sur un ensemble de n tâches à priorité fixe noté $\tau = \{\tau_1, ..., \tau_n\}$ et tel que, $\forall \tau_i \in \tau, P_i = i$. Hormis les n tâches précédentes, nous créons une tâche à priorité fixe, plus prioritaire que n'importe quelle tâche de τ, nommée *Schedule*, chargée du changement de contexte, qui intervient dès lors qu'une nouvelle tâche à priorité dynamique se voit attribuer le processeur. De plus, une fonction d'interruption de type ISR2, nommée *BaseTemps*, est déclenchée périodiquement à l'aide d'un compteur afin de gérer la base de temps ainsi que les activations des tâches de l'ensemble ρ. En outre, toutes les tâches de l'ensemble τ doivent être préemptives afin de laisser la tâche *Schedule* s'exécuter lorsqu'une nouvelle tâche activée par la fonction *BaseTemps* doit récupérer le processeur. Tout comme la fonction d'interruption *BaseTemps*, la tâche à priorité fixe *Schedule* peut donc préempter n'importe quelle tâche à priorité dynamique. Ainsi, les tâches à priorité dynamique ne peuvent être non préemptives qu'entre elles. A chacune de ses invocations, la fonction d'interruption *BaseTemps* active les tâches de l'ensemble ρ qui doivent l'être. Dès lors qu'au moins une tâche de ρ, possédant une priorité dynamique strictement inférieure à celle de la tâche *courante* notée P_{edf}, a été activée, la tâche *Schedule*, chargée d'attribuer le processeur à la tâche dont le rang dans la file d'attente lui est spécifié par la variable globale rg, est activée. Le rang rg correspond alors, si tant est qu'elle existe, à la tâche possédant la priorité dynamique, strictement inférieure à P_{edf}, la plus faible parmi les tâches venant d'être activées. L'algorithme 5.1 détaille le fonctionnement de la fonction d'interruption *BaseTemps*.

rg : **rang de l'éventuelle tâche élue** ; tmp : **structure *TachePrete*** ; i, j : **entiers** ;
P_{edf} : **priorité dynamique de la tâche courante** ;
P_{min} : **priorité dynamique** ; $file$: **file d'attente pour les tâches à l'état *prêt*** ;
$i = 0$; $P_{min} = P_{edf}$;
Tant que $(i < n)$ **faire**
  Si $(t == t_i^{act})$ **Alors**
    $t_i^{act} = t_i^{act} + T_i$; $tmp.id = i$; $tmp.f = F_i$; $tmp.P = t + D_i$;
    $j = file.Ajouter(tmp)$;//renvoie le rang de tmp dans la file
    Si $((t + D_i) < P_{min})$ **Alors**
      $rg = j$; $P_{min} = t + D_i$;
    Fin Si
  Fin Si
  $i = i + 1$;
Fait
$t = t + 1$;
Si $(P_{min} < P_{edf})$ **Alors**
  $ActivateTask(Schedule)$;
Fin Si

Algorithme 5.1: Algorithme de la fonction d'interruption *BaseTemps*

La tâche *Schedule* est chargée d'attribuer le processeur à la tâche dont le rang dans la file d'attente est spécifié par la variable globale rg. Dans le cas où la nouvelle tâche *courante* serait non préemptive, nous ramenons sa priorité dynamique à 0 jusqu'à la fin de son exécution afin qu'aucune autre tâche ne puisse la préempter. Soient P_{edf} la priorité dynamique de la tâche *courante* et τ_{fixe} la tâche à priorité fixe sur laquelle celle-ci s'exécute. La nouvelle tâche *courante* est exécutée via la tâche à priorité fixe τ_{fixe+1}. Aussi, à chaque élection, nous prenons soin de conserver l'ancienne valeur de P_{edf} dans une pile afin de la récupérer lorsque la nouvelle tâche élue aura terminé son exécution. L'algorithme 5.2 détaille le fonctionnement de la tâche *Schedule*.

rg : **rang de la tâche élue dans la file d'attente passé en argument** ;
P_{edf} : **entier signifiant la priorité dynamique de la tâche courante** ;
$pile$: **pile des priorités dynamiques des tâches ayant été élues** ;
$file$: **file d'attente pour les tâches à l'état *prêt*** ;
tmp : **structure *TachePrete*** ;
$tmp = file.Get(rg)$;
$file.Delete(rg)$;
$pile.Empile(P_{edf})$;
$\tau_{fixe} = \tau_{fixe+1}$;
Si $(tmp.f == true)$ **Alors**
  $P_{edf} = tmp.P$;
Sinon
  $P_{edf} = 0$;
Fin Si
$Associer(\tau_{fixe}, tmp.id)$;
$ActivateTask(\tau_{fixe})$;
$TerminateTask()$;

Algorithme 5.2: Algorithme de la tâche *Schedule*

Enfin, une fois que l'exécution d'une tâche ordonnancée avec EDF a été lancée à travers une tâche à priorité fixe, celle-ci s'achève obligatoirement en appelant une fonction nommée *FinTache*. Cette fonction a pour rôle de dépiler l'ancienne valeur P_{edf} et de terminer proprement la tâche en vérifiant dans la file d'attente si une tâche possède une priorité dynamique inférieure à celle venant d'être dépilée auquel cas la tâche *Schedule* serait activée. L'algorithme 5.3 détaille le fonctionnement de la fonction *FinTache*.

```
rg : rang de l'éventuelle tâche élue ;
P_edf : entier signifiant la priorité dynamique de la tâche courante ;
P_min : priorité dynamique minimale parmi les tâches activées et la courante ;
pile : pile des priorités dynamiques des tâches ayant été élues ;
file : file d'attente pour les tâches à l'état prêt ;
tmp : structure TachePrete ;
i : entier ;
P_edf = pile.Depile() ;
τ_fixe = τ_fixe−1 ;
P_min = P_edf ;
i = 0 ;
Tant que (i < file.Taille()) faire
    tmp = file.Get(i) ;
    Si (tmp.P < P_min) Alors
        rg = i ;
        P_min = tmp.P ;
    Fin Si
    i = i + 1 ;
Fait
Si (P_min < P_edf) Alors
    ChainTask(Schedule) ;
Sinon
    TerminateTask() ;
Fin Si
```

Algorithme 5.3: Algorithme de la fonction *FinTache*

5.a.ii Charges dues au noyau OSEK avec algorithme EDF

Rappelons que la fonction d'interruption *BaseTemps* est chargée d'entretenir la base de temps nécessaire à l'ordonnancement EDF et d'activer les tâches de l'ensemble ρ qui doivent l'être. Cette fonction apporte donc naturellement deux charges notées C_{tick}^{edf} et C_{act}^{edf}. La charge C_{act}^{edf} correspond à la durée d'exécution pire cas du code effectuant les activations, c'est à dire le code délimité par l'instruction conditionnelle "**Si** $(t == t_i^{act})$ **Alors ... Fin Si**" figurant à l'algorithme 5.1. En supposant qu'aucune tâche ne vérifie la condition précédente, le reste du code est systématiquement exécuté à chaque activation de la fonction *BaseTemps* et doit être comptabilisé dans la charge C_{tick}^{edf}. Nous notons T_{tick} la période de la fonction d'interruption *BaseTemps*. De plus, les changements de contexte, nécessaires au noyau pour démarrer et clôturer proprement la fonction d'interruption *BaseTemps*, sont inclus dans la charge C_{tick}^{edf}. Enfin, l'algorithme 5.1 se termine par un appel éventuel au service *ActivateTask*. Ce service permet l'activation de la tâche *Schedule* qui attribue le processeur à la tâche élue. L'exécution de la tâche *Schedule* amène une troisième charge notée C_{sched}^{edf}. Finalement, le coût du service *ActivateTask*, clôturant l'algorithme 5.1, est affecté à la charge C_{sched}^{edf}. Reste la fonction *FinTache* dont l'appel est obligatoire et nécessaire pour achever proprement toute tâche à priorité dynamique. La durée de son exécution produit une dernière charge notée C_{term}^{edf}. Nous rappelons que cette fonction vise à vérifier si, à l'occasion de la terminaison de la tâche courante, une tâche de la file d'attente doit être élue. Les services *TerminateTask* et *ChainTask*, présents à la fin de l'algorithme 5.3, sont respectivement imputés aux charges C_{term}^{edf} et C_{sched}^{edf}. Les quatre charges précédentes sont récapitulées au tableau 5.2. Nous précisons que pour chaque service *ActivateTask*, *ChainTask* ou *TerminateTask*, nous incluons dans son coût celui du changement de contexte qu'il engendre au niveau des tâches à priorité fixe. Nous soulignons que, parmi les quatre charges précédentes, seule la durée C_{act}^{edf} est constante, les autres varient linéairement avec n.

Symbole	*Description*
C_{tick}^{edf}	Durée d'exécution pire cas du code nécessaire à la gestion de la base de temps sur laquelle l'ordonnancement EDF est basé.
C_{act}^{edf}	Durée d'exécution pire cas du code permettant l'activation d'une tâche en la faisant passer de l'état *suspendu* à l'état *prêt*.
C_{sched}^{edf}	Durée d'exécution pire cas de la fonction de changement de contexte. Cette fonction intervient à chaque fois qu'une tâche se voit attribuer le processeur, celle-ci passe alors de l'état *prêt* à l'état *courant*.
C_{term}^{edf}	Durée d'exécution pire cas de la fonction *FinTache* appelée à la fin de l'exécution de toute tâche ordonnancée selon la politique EDF.

TABLE 5.2 – Description des différentes charges dues à l'algorithme EDF

5.b Règles pour le calcul du temps de réponse pire cas d'une tâche ordonnancée EDF

Nous récapitulons maintenant les règles à suivre durant le calcul du temps de réponse pire cas de toute tâche appartenant à un ensemble de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes, non concrètes et n'utilisant aucune ressource.

La politique d'ordonnancement EDF étant basée sur des priorités dynamiques, son scénario pire cas correspond à celui exposé au paragraphe 2.c.ii.2. Sachant que nous souhaitons intégrer les charges cumulées du noyau OSEK et de notre implémentation de la politique EDF, nous devons tenir compte du fait que, lorsqu'une tâche s'exécute, chaque activation d'une tâche moins prioritaire génère une charge C_{act}^{edf}. Afin de maximiser leur charge de travail et d'obtenir un scénario réellement pire cas, durant le calcul du temps de réponse pire cas d'une tâche ρ_i, les tâches moins prioritaires qu'elle doivent également être activées à la date 0. Finalement, lors du calcul de son temps de réponse pire cas, hormis la tâche ρ_i qui est activée à une date a telle que $a \in [0, T_i[$, toutes les tâches sont activées simultanément à la date 0.

Le scénario pire cas décrit au paragraphe 2.c.ii.2 impose, afin de trouver le temps de réponse pire cas d'une tâche ρ_i, de tester chacune des dates d'activation appartenant à l'ensemble A_i. Nous rappelons que l'ensemble A_i est défini comme suit :

$$A_i = \bigcup_{a \in [0, T_i[} A_i(a) \text{ avec } A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L_i(a) - C_i[\}$$

Sachant que, du fait qu'elle considère l'ensemble des tâches de l'application, la plus longue période d'activité L, étudiée au paragraphe 2.c.i.1, est toujours plus longue que n'importe quelle période d'activité de niveau de priorité $PG_i(t_i)$. Afin de ne pas devoir recalculer une nouvelle période d'activité pour chaque ensemble $A_i(a)$, nous allons, sans perdre de dates d'activation, redéfinir le sous ensemble $A_i(a)$ comme suit :

$$A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L - C_i[\}$$

Lors du calcul du temps de réponse pire cas d'une tâche ρ_i, ordonnancée avec notre implémentation de la politique EDF, nous suivrons le scénario suivant : Le temps de réponse pire cas d'une tâche ρ_i, appartenant à ρ, est obtenu lorsque toutes les autres tâches sont activées simultanément à la date 0. Ce temps de réponse pire cas correspond alors au temps de réponse maximum obtenu en testant toutes les dates d'activation de l'ensemble A_i défini comme suit :

$$A_i = \bigcup_{a \in [0, T_i[} A_i(a) \text{ avec } A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L - C_i[\}$$

L'intégration des charges C_{tick}^{edf}, C_{act}^{edf}, C_{sched}^{edf} et C_{term}^{edf}, durant le calcul du temps de réponse pire cas d'une tâche ρ_i, est identique à celle des charges C_{tick}, C_{act}, C_{sched} et C_{term} dans le cas d'un ordonnancement FP/FIFO. Nous nous reportons donc, sur ce point, aux règles énoncées à la sous section 4.b.iv. Nous soulignons que, cette fois, de part notre implémentation, l'exécution d'une tâche est toujours précédée d'une charge C_{sched}^{edf}. Ainsi, par rapport à la charge C_{sched}, le fait de comptabiliser une charge C_{sched}^{edf} pour chaque exécution ne constitue plus une simplification mais représente bien la réalité.

Ainsi, l'intégration des charges précédentes, au calcul de la plus longue période d'activité spécifié à la propriété 2.1, nous mène à la suite suivante :

$$\begin{cases} L^0 = \sum_{\rho_j \in \rho} (C_{act}^{edf} + C_{sched}^{edf} + C_j + C_{term}^{edf}) + C_{tick}^{edf} \\ L^{m+1} = \sum_{\rho_j \in \rho} \lceil \frac{L^m}{T_j} \rceil (C_{act}^{edf} + C_{sched}^{edf} + C_j + C_{term}^{edf}) + \lceil \frac{L^m}{T_{tick}} \rceil C_{tick}^{edf} \end{cases}$$

5.c Notations

Nous considérons un ensemble de n tâches périodiques $\rho = \{\rho_1, ..., \rho_n\}$ soumis à la politique d'ordonnancement EDF. Chaque tâche de l'ensemble ρ pouvant être préemptive ou non préemptive, nous notons ρ^{NP} le sous ensemble des tâches non préemptives appartenant à ρ. De plus, chaque tâche ρ_i étant caractérisée à l'aide des paramètres présentés à la sous section 5.a.i, nous définissons maintenant les sous ensembles de ρ suivants :

- $hp_i(t_i)$: L'ensemble des tâches, autres que ρ_i, ayant au moins une instance dont la priorité dynamique est inférieure ou égale à celle de la tâche ρ_i activée à la date t_i. Considérons une tâche ρ_j appartenant à l'ensemble $\rho - \{\rho_i\}$ et activée à la date t_j. La priorité dynamique de la tâche ρ_j vaut alors $t_j + D_j$. Cette priorité dynamique est inférieure ou égale à celle de la tâche ρ_i activée à la date t_i si, et seulement si, $t_j+D_j \leq t_i+D_i$. Ainsi, la tâche ρ_j possède une priorité dynamique inférieure ou égale à t_i+D_i lorsque celle-ci est activée à une date t_j telle que $t_j \leq t_i+D_i-D_j$. Autrement dit, seules les activations de la tâche ρ_j sur l'intervalle $[0, t_i + D_i - D_j]$ mènent à une priorité dynamique inférieure ou égale à $t_i + D_i$. A l'évidence, l'intervalle précédent n'existe que si $D_j \leq t_i + D_i$. Ainsi, l'ensemble $hp_i(t_i)$, qui exclut la tâche ρ_i, se définit comme suit : $hp_i(t_i) = \{\rho_j \in \rho - \{\rho_i\}/D_j \leq (t_i + D_i)\}$.
- $lp_i(t_i)$: L'ensemble des tâches dont toutes les priorités dynamiques sont strictement supérieures à celle de la tâche ρ_i activée à la date t_i. Comme nous l'avons expliqué ci-dessus, une tâche ρ_j appartenant à l'ensemble ρ ne peut obtenir de priorité dynamique inférieure ou égale à t_i+D_i que si $D_j \leq t_i+D_i$. Dans le cas contraire, toute instance de la tâche ρ_j possède une priorité dynamique strictement supérieure à t_i+D_i. Ainsi, l'ensemble $lp_i(t_i)$ se définit comme suit : $lp_i(t_i) = \{\rho_j \in \rho/D_j > (t_i + D_i)\}$. Notez que la tâche ρ_i en est naturellement exclue.
- $lp_i^{NP}(t_i)$: L'ensemble des tâches non préemptives dont toutes les priorités dynamiques sont strictement supérieures à celle de la tâche ρ_i activée à la date t_i. Là encore, la tâche ρ_i est exclue de cet ensemble qui se définit comme suit : $lp_i^{NP}(t_i) = \{\rho_j \in \rho^{NP}/D_j > (t_i + D_i)\}$.

Nous définissons la fonction $s_i(t)$ permettant de calculer la date de la première activation, dans l'intervalle $[0, T_i[$, d'une tâche τ_i activée à la date t :

$$s_i(t) = t - \lfloor \frac{t}{T_i} \rfloor T_i$$

Nous précisons que, dans notre implémentation, les paramètres T_i et D_i de toute tâche ρ_i sont directement exprimés en nombre de périodes T_{tick}. Ainsi, ces paramètres correspondent bien à la réalité. Nous n'aurons donc, dans l'analyse présentée aux sections 5.d et 5.e, aucun terme T_i^* comme celui défini à la sous section 4.c.i.

5.d Temps de réponse pire cas d'une tâche préemptive

Cette section détaille le calcul du temps de réponse pire cas d'une tâche périodique préemptive. Soit ρ_i une tâche préemptive d'un ensemble de tâches périodiques ρ. Le temps de réponse de la tâche ρ_i activée à la date t se note $r_{i,t}$. Comme nous le disons à la section 5.b, le temps de réponse pire cas r_i de la tâche ρ_i correspond à son temps de réponse maximum observé sur l'ensemble des dates d'activation appartenant à A_i. Ainsi, le temps de réponse pire cas de cette tâche est donné par la formule :

$$r_i = \max_{t \in A_i}(r_{i,t})$$

$$\text{Avec } A_i = \bigcup_{a \in [0, T_i[} A_i(a) \text{ avec } A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L - C_i[\}$$

A la sous section 5.d.i, nous expliquons le calcul du temps de réponse $r_{i,t}$ de la tâche ρ_i activée à la date t. Lors de ce calcul, nous prenons soin de respecter le scénario pire cas, décrit à la section 5.b, en considérant que les autres tâches sont toujours activées simultanément à la date 0 alors que la première activation de la tâche ρ_i intervient à la date $a \in [0, T_i[$. Une fois le temps de réponse pire cas de la tâche ρ_i calculé, nous présentons, à la sous section 5.d.ii, les équations nécessaires à la détermination des temps passés à exécuter chaque charge cumulée du noyau OSEK et de l'algorithme EDF sur ce temps de réponse pire cas.

5.d.i Calcul du temps de réponse de la tâche ρ_i activée à la date t

Nous considérons une tâche préemptive ρ_i appartenant à l'ensemble de n tâches périodiques ρ et activée, durant la plus longue période d'activité L, à la date $t = a + k \times T_i$ où $k \in N$ et $a \in [0, T_i[$. Nous soulignons que la première activation de la tâche ρ_i intervient à la date a. Le calcul de son temps de réponse nécessite le calcul de sa date de fin d'exécution $\omega_{i,t}$. Nous commençons donc par calculer la date $\omega_{i,t}$ en tenant compte de l'ensemble des charges dues au système d'exploitation et à l'ordonnancement EDF ainsi que de l'éventuel retard dû à une tâche moins prioritaire non préemptive.

Sachant que la tâche ρ_i est activée à la date t, toute tâche ρ_j de l'ensemble $hp_i(t)$ possède une priorité dynamique inférieure ou égale à ρ_i à chacune de ses activations dans l'intervalle $[0, t + D_i - D_j]$. Autrement dit, seules $1 + \lfloor \frac{t+D_i-D_j}{T_j} \rfloor$ activations de la tâche ρ_j pourront préempter la tâche ρ_i activée à la date t. De plus, l'exécution de cette activation de la tâche ρ_i s'achève à la date $\omega_{i,t}$. Ainsi, au plus, $\lceil \frac{\omega_{i,t}}{T_j} \rceil$ activations de la tâche ρ_j doivent être prises en compte dans le calcul de $\omega_{i,t}$. Qui plus est, toute tâche de $hp_i(t)$ doit se voir attribuer le processeur avant de débuter son exécution puis terminer celle-ci en appelant la fonction *FinTache*. Ainsi, lors du calcul de la date $\omega_{i,t}$, la durée requise pour exécuter l'ensemble des tâches de $hp_i(t)$ vaut :

$$\sum_{\tau_j \in hp_i(t)} \min(1 + \lfloor \frac{t+D_i-D_j}{T_j} \rfloor, \lceil \frac{\omega_{i,t}}{T_j} \rceil)(C_{sched}^{edf} + C_j + C_{term}^{edf})$$

Concernant la tâche ρ_i, celle-ci étant activée à la date t, seules ses activations sur l'intervalle $[s_i(t), t]$ possèdent une priorité dynamique inférieure ou égale à $t + D_i$. Ainsi, seules $1 + \lfloor \frac{t}{T_i} \rfloor$ activations de la tâche ρ_i peuvent entrer en considération dans le calcul de la date

$\omega_{i,t}$. De plus, seules $\lceil \frac{\max(\omega_{i,t}-s_i(t),0)}{T_i} \rceil$ activations de cette tâche se produisent sur l'intervalle $[s_i(t), \omega_{i,t}[$. A chacune de ses activations, la tâche ρ_i doit se voir attribuer le processeur avant de débuter son exécution puis terminer celle-ci en appelant la fonction *FinTache*. Finalement, la contribution de la tâche ρ_i, durant le calcul de la date $\omega_{i,t}$, revient à :

$$\min(1 + \lfloor \frac{t}{T_i} \rfloor, \lceil \frac{\max(\omega_{i,t}-s_i(t),0)}{T_i} \rceil)(C_{sched}^{edf} + C_i + C_{term}^{edf})$$

Hormis ρ_i, toutes les tâches de l'ensemble ρ sont activées périodiquement sur l'intervalle $[0, \omega_{i,t}[$ ce qui apporte le terme suivant :

$$\sum_{\rho_j \in \rho - \{\rho_i\}} \lceil \frac{\omega_{i,t}}{T_j} \rceil C_{act}^{edf}$$

La tâche ρ_i, quant à elle, est activée périodiquement sur l'intervalle $[s_i(t), \omega_{i,t}[$ ce qui s'écrit : $\lceil \frac{\max(\omega_{i,t}-s_i(t),0)}{T_i} \rceil C_{act}^{edf}$

Nous ajoutons maintenant le temps passé à gérer la base de temps sur l'intervalle $[0, \omega_{i,t}[$:

$$\lceil \frac{\omega_{i,t}}{T_{tick}} \rceil C_{tick}^{edf}$$

L'effet non préemptif, expliqué à la sous section 2.a.iii, correspond au fait qu'une tâche non préemptive puisse retarder l'exécution d'une tâche plus prioritaire. Ainsi, si l'ensemble $lp_i^{NP}(t)$ n'est pas vide, nous devons considérer cet effet non préemptif. Pour ce faire, nous recherchons dans l'ensemble $lp_i^{NP}(t)$ la tâche possédant la plus longue durée d'exécution pire cas. Une fois activée, cette tâche ne monopolise le processeur qu'à partir du moment où celui-ci lui a été attribué. Nous supposons donc que la charge C_{sched} lui permettant d'obtenir le processeur débute son exécution un cycle avant la date 0 et obtenons le terme suivant :

$$\max_{\rho_k \in lp_i^{NP}(t)} (C_{sched}^{edf} + C_k + C_{term}^{edf} - 1)$$

Finalement, lorsqu'une tâche préemptive ρ_i est activée à la date $t \in [0, L - C_i[$, sa date de fin d'exécution $\omega_{i,t}$ coïncide avec le plus petit point fixe de la suite ci-dessous. La complexité du calcul de $\omega_{i,t}$ est en $O(n.L)$.

$$\begin{cases} \omega_{i,t}^0 = \sum_{\tau_j \in hp_i(t)} (C_{sched}^{edf} + C_j + C_{term}^{edf}) + \eta_{i,t}(C_{act}^{edf} + C_{sched}^{edf} + C_i + C_{term}^{edf}) + \\ \sum_{\rho_j \in \rho - \{\rho_i\}} C_{act}^{edf} + C_{tick}^{edf} + \max_{\rho_k \in lp_i^{NP}(t)} (C_{sched}^{edf} + C_k + C_{term}^{edf} - 1) \\ \omega_{i,t}^{m+1} = \sum_{\tau_j \in hp_i(t)} \min(1 + \lfloor \frac{t+D_i-D_j}{T_j} \rfloor, \lceil \frac{\omega_{i,t}^m}{T_j} \rceil)(C_{sched}^{edf} + C_j + C_{term}^{edf}) + \\ \min(1 + \lfloor \frac{t}{T_i} \rfloor, \lceil \frac{\max(\omega_{i,t}^m - s_i(t),0)}{T_i} \rceil)(C_{sched}^{edf} + C_i + C_{term}^{edf}) + \sum_{\rho_j \in \rho - \{\rho_i\}} \lceil \frac{\omega_{i,t}^m}{T_j} \rceil C_{act}^{edf} + \\ \lceil \frac{\max(\omega_{i,t}^m - s_i(t),0)}{T_i} \rceil C_{act}^{edf} + \lceil \frac{\omega_{i,t}^m}{T_{tick}} \rceil C_{tick}^{edf} + \max_{\rho_k \in lp_i^{NP}(t)} (C_{sched}^{edf} + C_k + C_{term}^{edf} - 1) \end{cases}$$

$$\text{où } \eta_{i,t} = \begin{cases} 1 \text{ si } s_i(t) = 0 \\ 0 \text{ autrement} \end{cases}$$

Si $\omega_{i,t}$ s'avère être strictement inférieur à t, cela signifie que l'instance de la tâche ρ_i activée à la date t n'appartient pas à la plus longue période d'activité de niveau de priorité $PG_i(t)$. Dans ce cas, par défaut, le temps de réponse de la tâche ρ_i vaut $r_{i,t} = C_{act}^{edf} + C_{sched}^{edf} + C_i + C_{term}^{edf}$. Autrement, ce temps de réponse est égal à $\omega_{i,t} - t$. Nous aboutissons donc à la formule suivante :

$$r_{i,t} = \max(C_{act}^{edf} + C_{sched}^{edf} + C_i + C_{term}^{edf}, \omega_{i,t} - t)$$

5.d.ii Temps consacré à chaque charge du noyau sur le temps de réponse pire cas de la tâche ρ_i

Une fois r_i connu, nous pouvons calculer, à titre purement indicatif, les durées d'exécution de chaque charge cumulée du système d'exploitation et de l'algorithme EDF sur ce temps de réponse pire cas. Lors de la recherche de r_i, nous prendrons soin de noter la date d'activation de la tâche ρ_i, notée t_i^{wcrt}, qui mène à ce temps de réponse pire cas, ainsi que la date de la fin de l'exécution associée à cette activation, notée ω_i^{wcrt}. Notez que lorsqu'une tâche non préemptive est activée à une date t, le temps de réponse correspondant est également noté $r_{i,t}$. Le calcul de ce dernier est détaillé à la section 5.e.

La durée totale d'exécution de la charge C_{act}^{edf}, sur l'intervalle $[t_i^{wcrt}, t_i^{wcrt} + r_i[$, provient des activations périodiques de chaque tâche de l'ensemble ρ. Pour chaque tâche ρ_j de l'ensemble ρ, différente de ρ_i, nous tenons compte du décalage entre la date d'activation t_i^{wcrt} de la tâche ρ_i et la première activation de la tâche ρ_j après cette date. Ce décalage est noté d_j. La date t_i^{wcrt} correspondant à une activation de la tâche ρ_i, aucun décalage n'existe pour celle-ci. Nous précisons que la complexité du calcul de r_i^{act} est en $O(n)$.

$$r_i^{act} = \sum_{\rho_j \in \rho - \{\rho_i\}} \lceil \frac{\max(r_i - d_j, 0)}{T_j} \rceil C_{act}^{edf} + \lceil \frac{r_i}{T_i} \rceil C_{act}^{edf} \text{ où } d_j = \lceil \frac{t_i^{wcrt}}{T_j} \rceil T_j - t_i^{wcrt}$$

La durée totale d'exécution de la charge C_{tick}, sur le même intervalle, découle directement de ses exécutions périodiques de période T_{tick}. Là encore, nous tenons compte du décalage, noté d_{tick}, entre la date d'activation t_i^{wcrt} de la tâche ρ_i et la première exécution de la charge C_{tick} après cette date. Nous précisons que la complexité du calcul de r_i^{tick} est en $O(1)$.

$$r_i^{tick} = \lceil \frac{\max(r_i - d_{tick}, 0)}{T_{tick}} \rceil C_{tick}^{edf} \text{ où } d_{tick} = \lceil \frac{t_i^{wcrt}}{T_{tick}} \rceil T_{tick} - t_i^{wcrt}$$

Lors du calcul de ω_i^{wcrt}, nous avons intégré les exécutions et les terminaisons de toutes les tâches appartenant à $hp_i(t_i^{wcrt}) \cup \{\rho_i\}$. Pour chaque tâche ρ_j de l'ensemble $hp_i(t_i^{wcrt})$, nous avons considéré l'ensemble de ses activations sur l'intervalle $[0, t_i^{wcrt} + D_i - D_j] \cap [0, \omega_i^{wcrt}[$. Concernant la tâche ρ_i, nous avons comptabilisé ses activations sur l'intervalle $[s_i(t_i^{wcrt}), t_i^{wcrt}] \cap [s_i(t_i^{wcrt}), \omega_i^{wcrt}[$. Sachant que $\omega_i^{wcrt} > t_i^{wcrt}$, l'intervalle précédent équivaut à $[s_i(t_i^{wcrt}), t_i^{wcrt}]$. Pour chacune des activations précédentes, comme nous l'avons fait au paragraphe 4.c.iv.2, nous déterminons si la charge C_{term}^{edf} s'exécute dans l'intervalle $[t_i^{wcrt}, \omega_i^{wcrt}[$ et si cette exécution est totale ou partielle. De plus, si la tâche ρ_i est retardée par une tâche non préemptive moins prioritaire, c'est à dire si l'ensemble $lp_i^{NP}(t)$ n'est pas vide, nous devons considérer la charge C_{term}^{edf} qui termine l'exécution de cette tâche moins prioritaire. Nous précisons que la complexité du calcul de r_i^{term} est en $O(n^2.L)$.

$$r_i^{term} = \sum_{\rho_j \in hp_i(t_i^{wcrt})} \sum_{t \in [0, t_i^{wcrt} + D_i - D_j] \cap [0, \omega_i^{wcrt}[} \max(\min(r_{j,t} + t - t_i^{wcrt}, C_{term}^{edf}), 0) +$$

$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \max(\min(r_{i,t} + t - t_i^{wcrt}, C_{term}^{edf}), 0) + \gamma_i(t_i^{wcrt}) \max(\min(t_i^{NP} - t_i^{wcrt}, C_{term}^{edf}), 0)$$

$$\text{où } \gamma_i(t_i^{wcrt}) = \begin{cases} 1 \text{ si } lp_i^{NP}(t_i^{wcrt}) \neq \phi \\ 0 \text{ autrement} \end{cases} \text{ et } t_i^{NP} = \max_{\rho_k \in lp_i^{NP}(t_i^{wcrt})} (C_{sched}^{edf} + C_k + C_{term}^{edf} - 1)$$

Enfin, chaque charge C_{term}^{edf} prise en compte durant le calcul de la date ω_i^{wcrt} est obligatoirement précédée par l'exécution de la tâche associée ainsi que par une charge C_{sched}^{edf}. C'est pourquoi, le calcul de la durée totale d'exécution de la charge C_{sched}^{edf}, sur l'intervalle $[t_i^{wcrt}, t_i^{wcrt} + r_i[$ est conforme au calcul précédent à cela prêt que, pour chaque instance, nous déduisons une durée C_{term}^{edf} et la durée d'exécution pire cas de la tâche correspondante. De plus, si la tâche ρ_i est retardée par une tâche non préemptive moins prioritaire, nous devons considérer la charge C_{sched}^{edf} qui précède l'exécution de cette tâche moins prioritaire. Nous précisons que la complexité du calcul de r_i^{sched} est en $O(n^2.L)$.

$$r_i^{sched} = \sum_{\rho_j \in hp_i(t_i^{wcrt})} \sum_{t \in [0, t_i^{wcrt} + D_i - D_j] \cap [0, \omega_i^{wcrt}[} \max(\min(r_{j,t} + t - C_{term}^{edf} - C_j - t_i^{wcrt}, C_{sched}^{edf}), 0) +$$
$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \max(\min(r_{i,t} + t - C_{term}^{edf} - C_i - t_i^{wcrt}, C_{sched}^{edf}), 0) +$$
$$\gamma_i(t_i^{wcrt}) \max(C_{sched}^{edf} - 1 - t_i^{wcrt}, 0)$$
$$\text{où } \gamma_i(t_i^{wcrt}) = \begin{cases} 1 \text{ si } lp_i^{NP}(t_i^{wcrt}) \neq \phi \\ 0 \text{ autrement} \end{cases}$$

5.e Temps de réponse pire cas d'une tâche non préemptive

Cette section détaille le calcul du temps de réponse pire cas d'une tâche périodique non préemptive. Soit ρ_i une tâche non préemptive d'un ensemble de tâches périodiques ρ. Le temps de réponse de la tâche ρ_i activée à la date t se note $r_{i,t}$. Comme nous le disons à la section 5.b, le temps de réponse pire cas r_i de la tâche ρ_i correspond à son temps de réponse maximum observé sur l'ensemble des dates d'activation appartenant à A_i. Ainsi, le temps de réponse pire cas de cette tâche est donné par la formule :

$$r_i = \max_{t \in A_i}(r_{i,t})$$
$$\text{Avec } A_i = \bigcup_{a \in [0, T_i[} A_i(a) \text{ avec } A_i(a) = \{a_i^k = a + k \times T_i, k \in N / a_i^k \in [0, L - C_i[\}$$

A la sous section 5.e.i, nous expliquons le calcul du temps de réponse $r_{i,t}$ de la tâche ρ_i activée à la date t. Lors de ce calcul, nous prenons soin de respecter le scénario pire cas, décrit à la section 5.b, en considérant que les autres tâches sont toujours activées simultanément à la date 0 alors que la première activation de la tâche ρ_i intervient à la date $a \in [0, T_i[$. Une fois le temps de réponse pire cas de la tâche ρ_i calculé, nous présentons, à la sous section 5.e.ii, les équations nécessaires à la détermination des temps passés à exécuter chaque charge cumulée du noyau et de l'algorithme EDF sur ce temps de réponse pire cas.

5.e.i Calcul du temps de réponse de la tâche ρ_i activée à la date t

Nous considérons donc une tâche non préemptive ρ_i activée, durant la plus longue période d'activité L, à la date $t = a + k \times T_i$ où $k \in N$ et $a \in [0, T_i[$. Le calcul de son temps de réponse nécessite le calcul de sa date de début d'exécution $\overline{\omega}_{i,t}$. Dès son exécution entamée, la tâche ρ_i ne peut plus être interrompue par une autre tâche de l'ensemble ρ. Cependant, entre son activation à la date t et le début de son exécution à la date $\overline{\omega}_{i,t}$, les tâches plus prioritaires exploitent le processeur. Nous commençons donc par calculer la date $\overline{\omega}_{i,t}$ puis la durée réelle de l'exécution qui s'en suit notée $z_{i,t}$.

Sachant que la tâche ρ_i est activée à la date t, toute tâche ρ_j de l'ensemble $hp_i(t)$ possède une priorité dynamique inférieure ou égale à ρ_i, activée à la date t, à chacune de ses activations dans l'intervalle $[0, t+D_i-D_j]$. Autrement dit, seules $1+\lfloor\frac{t+D_i-D_j}{T_j}\rfloor$ activations de la tâche ρ_j pourront retarder la tâche ρ_i activée à la date t. De plus, l'exécution de cette activation de la tâche ρ_i débute immédiatement après la date $\overline{\omega}_{i,t}$. Ainsi, au plus, $1+\lfloor\frac{\overline{\omega}_{i,t}}{T_j}\rfloor$ activations de la tâche ρ_j sont à considérer durant le calcul de $\overline{\omega}_{i,t}$. De plus, toute tâche de $hp_i(t)$ doit se voir attribuer le processeur avant de débuter son exécution puis terminer celle-ci en appelant la fonction *FinTache*. Ainsi, lors du calcul de la date $\overline{\omega}_{i,t}$, la durée requise pour exécuter l'ensemble des tâches de $hp_i(t)$ vaut :

$$\begin{aligned}&\sum_{\tau_j \in hp_i(t)} \min(1+\lfloor\tfrac{t+D_i-D_j}{T_j}\rfloor, 1+\lfloor\tfrac{\overline{\omega}_{i,t}}{T_j}\rfloor)(C^{edf}_{sched}+C_j+C^{edf}_{term})\\ &= \sum_{\tau_j \in hp_i(t)} (1+\lfloor\tfrac{\min(t+D_i-D_j,\overline{\omega}_{i,t})}{T_j}\rfloor)(C^{edf}_{sched}+C_j+C^{edf}_{term})\end{aligned}$$

Concernant la tâche ρ_i, celle-ci étant activée à la date t, seules ses activations sur l'intervalle $[s_i(t), t]$ possèdent une priorité dynamique inférieure ou égale à $t+D_i$. Pour autant, seules ses activations sur l'intervalle $[s_i(t), t[$ doivent être prises en compte dans le calcul de la date $\overline{\omega}_{i,t}$, car l'exécution de la tâche ρ_i activée à la date t se déroule après cette date. Ainsi, seules $\lfloor\frac{t}{T_i}\rfloor$ activations de la tâche ρ_i peuvent entrer en considération dans le calcul de la date $\overline{\omega}_{i,t}$. De plus, seules ses activations se produisant sur l'intervalle $[s_i(t), \overline{\omega}_{i,t}]$ interviennent dans ce calcul. A chacune de ses activations, la tâche ρ_i doit se voir attribuer le processeur avant de débuter son exécution puis terminer celle-ci en appelant la fonction *FinTache*. Finalement, la contribution de la tâche ρ_i, durant le calcul de la date $\overline{\omega}_{i,t}$, revient à :

$$\min(\lfloor\tfrac{t}{T_i}\rfloor, \mu_{i,t}(\overline{\omega}_{i,t}))(C^{edf}_{sched}+C_i+C^{edf}_{term}) \text{ où}$$
$$\mu_{i,t}(\overline{\omega}_{i,t}) = \begin{cases} 1+\lfloor\frac{\overline{\omega}_{i,t}-s_i(t)}{T_i}\rfloor \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \text{ et } 0 \text{ autrement}\end{cases}$$

Hormis ρ_i, toutes les tâches de l'ensemble ρ sont activées périodiquement sur l'intervalle $[0, \overline{\omega}_{i,t}]$ ce qui apporte le terme suivant :

$$\sum_{\rho_j \in \rho - \{\rho_i\}} (1+\lfloor\tfrac{\overline{\omega}_{i,t}}{T_j}\rfloor)C^{edf}_{act}$$

La tâche ρ_i, quant à elle, est activée périodiquement sur l'intervalle $[s_i(t), \overline{\omega}_{i,t}]$ ce qui s'écrit :

$$\mu_{i,t}(\overline{\omega}_{i,t})C^{edf}_{act} \text{ où } \mu_{i,t}(\overline{\omega}_{i,t}) = \begin{cases} 1+\lfloor\frac{\overline{\omega}_{i,t}-s_i(t)}{T_i}\rfloor \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \text{ et } 0 \text{ autrement}\end{cases}$$

Nous ajoutons maintenant le temps passé à gérer la base de temps sur l'intervalle $[0, \overline{\omega}_{i,t}]$:

$$(1+\lfloor\tfrac{\overline{\omega}_{i,t}}{T_{tick}}\rfloor)C^{edf}_{tick}$$

Comme dans le cas d'une tâche préemptive, si l'ensemble $lp^{NP}_i(t)$ n'est pas vide, nous devons considérer l'effet non préemptif dû à la tâche de $lp^{NP}_i(t)$ possédant la durée d'exécution pire cas la plus longue.

$$\max_{\rho_k \in lp^{NP}_i(t)} (C^{edf}_{sched}+C_k+C^{edf}_{term}-1)$$

Finalement, lorsqu'une tâche non préemptive ρ_i est activée à la date $t \in [0, L - C_i[$, sa date de début d'exécution $\overline{\omega}_{i,t}$ coïncide avec le plus petit point fixe de la suite ci-dessous. Nous précisons que la complexité du calcul de $\overline{\omega}_{i,t}$ est en $O(n.L)$.

$$\begin{cases} \overline{\omega}^0_{i,t} = \sum\limits_{\tau_j \in hp_i(t)} (C^{edf}_{sched} + C_j + C^{edf}_{term}) + \eta_{i,t}(C^{edf}_{act} + C^{edf}_{sched} + C_i + C^{edf}_{term}) + \\ \sum\limits_{\rho_j \in \rho - \{\rho_i\}} C^{edf}_{act} + C^{edf}_{tick} + \max\limits_{\rho_k \in lp^{NP}_i(t)} (C^{edf}_{sched} + C_k + C^{edf}_{term} - 1) \\ \overline{\omega}^{m+1}_{i,t} = \sum\limits_{\tau_j \in hp_i(t)} (1 + \lfloor \frac{\min(t + D_i - D_j, \overline{\omega}^m_{i,t})}{T_j} \rfloor)(C^{edf}_{sched} + C_j + C^{edf}_{term}) + \\ \min(\lfloor \frac{t}{T_i} \rfloor, \mu_{i,t}(\overline{\omega}^m_{i,t}))(C^{edf}_{sched} + C_i + C^{edf}_{term}) + \sum\limits_{\rho_j \in \rho - \{\rho_i\}} (1 + \lfloor \frac{\overline{\omega}^m_{i,t}}{T_j} \rfloor) C^{edf}_{act} + \\ \mu_{i,t}(\overline{\omega}^m_{i,t}) C^{edf}_{act} + (1 + \lfloor \frac{\overline{\omega}^m_{i,t}}{T_{tick}} \rfloor) C^{edf}_{tick} + \max\limits_{\rho_k \in lp^{NP}_i(t)} (C^{edf}_{sched} + C_k + C^{edf}_{term} - 1) \end{cases}$$

$$\text{où } \eta_{i,t} = \begin{cases} 1 \text{ si } s_i(t) = 0 \\ 0 \text{ autrement} \end{cases} \text{ et } \mu_{i,t}(\overline{\omega}_{i,t}) = \begin{cases} 1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T_i} \rfloor \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \text{ et } 0 \text{ autrement} \end{cases}$$

Maintenant que nous connaissons la date après laquelle la tâche non préemptive ρ_i, activée à la date t, débute son exécution, nous nous intéressons à la durée de celle-ci notée $z_{i,t}$. L'activation de la tâche ρ_i à la date t ayant déjà été prise en compte lors du calcul de la date $\overline{\omega}_{i,t}$, nous ne devons plus considérer que son exécution ainsi que les charges C^{edf}_{sched} et C^{edf}_{term} représentant respectivement son élection et sa terminaison :

$$C^{edf}_{sched} + C_i + C^{edf}_{term}$$

Durant son exécution, la tâche ρ_i ne peut en aucun cas être préemptée par une autre tâche de l'ensemble ρ. Cependant, notre ordonnanceur EDF continue d'activer les tâches, au fur et à mesure, en vue de leurs prochaines exécutions. La prise en compte de ces activations nécessite de déterminer, pour chaque tâche ρ_j de l'ensemble ρ, le décalage, noté t^0_j, entre la date $\overline{\omega}_{i,t}$ et sa première activation après celle-ci. La prise en compte des activations des tâches de l'ensemble ρ, pendant l'exécution de la tâche ρ_i, correspond alors au terme suivant :

$$\sum_{\rho_j \in \rho} \lceil \frac{\max(z_{i,t} - t^0_j, 0)}{T_j} \rceil C^{edf}_{act} \text{ où } t^0_{j \neq i} = (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_j} \rfloor) T_j - \overline{\omega}_{i,t} \text{ et}$$

$$t^0_i = \begin{cases} (1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T_i} \rfloor) T_i - (\overline{\omega}_{i,t} - s_i(t)) \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \\ s_i(t) - \overline{\omega}_{i,t} \text{ autrement} \end{cases}$$

Enfin, nous intégrons le fait que, malgré l'exécution de la tâche ρ_i, l'ordonnanceur EDF doit maintenir sa base de temps. Là encore, nous tenons compte du décalage entre la date $\overline{\omega}_{i,t}$ et la première arrivée de la charge C_{tick} après cette date.

$$\lceil \frac{\max(z_{i,t} - t^0_{tick}, 0)}{T_{tick}} \rceil C^{edf}_{tick} \text{ où } t^0_{tick} = (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_{tick}} \rfloor) T_{tick} - \overline{\omega}_{i,t}$$

Ainsi, la durée de l'exécution de la tâche ρ_i, activée à la date t, coïncide avec le plus petit point fixe de la suite ci-dessous. La complexité du calcul de $z_{i,t}$ est en $O(n.L)$.

$$\begin{cases} z_{i,t}^0 = C_{sched}^{edf} + C_i + C_{term}^{edf} \\ z_{i,t}^{m+1} = C_{sched}^{edf} + C_i + C_{term}^{edf} + \sum_{\rho_j \in \rho} \lceil \frac{\max(z_{i,t}^m - t_j^0, 0)}{T_j} \rceil C_{act}^{edf} + \lceil \frac{\max(z_{i,t}^m - t_{tick}^0, 0)}{T_{tick}} \rceil C_{tick}^{edf} \end{cases}$$

où $t_{j\neq i}^0 = (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_j} \rfloor) T_j - \overline{\omega}_{i,t}$, $t_i^0 = \begin{cases} (1 + \lfloor \frac{\overline{\omega}_{i,t} - s_i(t)}{T_i} \rfloor) T_i - (\overline{\omega}_{i,t} - s_i(t)) \text{ si } \overline{\omega}_{i,t} \geq s_i(t) \\ s_i(t) - \overline{\omega}_{i,t} \text{ autrement} \end{cases}$ et

$$t_{tick}^0 = (1 + \lfloor \frac{\overline{\omega}_{i,t}}{T_{tick}} \rfloor) T_{tick} - \overline{\omega}_{i,t}$$

Si $\overline{\omega}_{i,t} + z_{i,t}$ s'avère être strictement inférieur à t, cela signifie que l'instance de la tâche ρ_i activée à la date t n'appartient pas à la plus longue période d'activité de niveau de priorité $PG_i(t)$. Dans ce cas, par défaut, le temps de réponse de la tâche ρ_i vaut $r_{i,t} = C_{act}^{edf} + C_{sched}^{edf} + C_i + C_{term}^{edf}$. Autrement, ce temps de réponse est égal à $\overline{\omega}_{i,t} + z_{i,t} - t$. Nous aboutissons donc à la formule suivante :

$$r_{i,t} = \max(C_{act}^{edf} + C_{sched}^{edf} + C_i + C_{term}^{edf}, \overline{\omega}_{i,t} + z_{i,t} - t)$$

5.e.ii Temps consacré à chaque charge du noyau sur le temps de réponse pire cas de la tâche ρ_i

Nous noterons t_i^{wcrt} la date d'activation qui mène au temps de réponse pire cas de la tâche ρ_i. La date à laquelle la tâche ρ_i débute son exécution et la durée de celle-ci, quand ρ_i est activée à la date t_i^{wcrt}, seront respectivement notées $\overline{\omega}_i^{wcrt}$ et z_i^{wcrt}. Comme nous l'avons fait à la sous section 5.d.ii dans le cas d'une tâche préemptive, nous calculons, toujours à titre purement indicatif, sur le r_i de la tâche non préemptive ρ_i, les durées d'exécution de chaque charge cumulée du système d'exploitation et de l'algorithme EDF. Nous rappelons que le temps de réponse d'une tâche préemptive ρ_i, activée à une date t, se note également $r_{i,t}$ et que son calcul est détaillé à la section 5.d.

Comme dans le cas d'une tâche préemptive, la durée totale d'exécution de la charge C_{act}^{edf}, sur l'intervalle $[t_i^{wcrt}, t_i^{wcrt} + r_i[$, provient des activations périodiques de chaque tâche de l'ensemble ρ. De même, la durée totale d'exécution de la charge C_{tick}^{edf}, sur le même intervalle, découle, là encore, directement de ses exécutions périodiques de période T_{tick}. Concernant les durées r_i^{act} et r_i^{tick}, nous nous reportons donc aux formules fournies à la sous section 5.d.ii.

Lors du calcul de $r_{i,t_i^{wcrt}}$, nous avons intégré les exécutions et les terminaisons de toutes les tâches appartenant à $hp_i(t_i^{wcrt}) \cup \{\rho_i\}$. Pour chaque tâche ρ_j de l'ensemble $hp_i(t_i^{wcrt})$, nous avons considéré l'ensemble de ses activations sur l'intervalle $[0, t_i^{wcrt} + D_i - D_j] \cap [0, \overline{\omega}_i^{wcrt}]$. Concernant la tâche ρ_i, nous avons comptabilisé ses activations sur l'intervalle $[s_i(t_i^{wcrt}), t_i^{wcrt}] \cap [s_i(t_i^{wcrt}), \overline{\omega}_i^{wcrt}]$. Sachant que $\overline{\omega}_i^{wcrt} > t_i^{wcrt}$, l'intervalle précédent équivaut à $[s_i(t_i^{wcrt}), t_i^{wcrt}]$. Pour chacune des activations précédentes, nous devons déterminer si la charge C_{term}^{edf} s'exécute dans l'intervalle $[t_i^{wcrt}, t_i^{wcrt} + r_i[$ et si cette exécution est totale

ou partielle. De plus, si la tâche ρ_i est retardée par une tâche non préemptive moins prioritaire, c'est à dire si l'ensemble $lp_i^{NP}(t)$ n'est pas vide, nous devons considérer la charge C_{term}^{edf} qui termine l'exécution de cette tâche moins prioritaire. Nous précisons que la complexité du calcul de r_i^{term} est en $O(n^2.L)$.

$$r_i^{term} = \sum_{\rho_j \in hp_i(t_i^{wcrt})} \sum_{t \in [0, t_i^{wcrt} + D_i - D_j] \cap [0, \overline{\omega}_i^{wcrt}]} \max(\min(r_{j,t} + t - t_i^{wcrt}, C_{term}^{edf}), 0) +$$
$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \max(\min(r_{i,t} + t - t_i^{wcrt}, C_{term}^{edf}), 0) + \gamma_i(t_i^{wcrt}) \max(\min(t_i^{NP} - t_i^{wcrt}, C_{term}^{edf}), 0)$$
$$\text{où } \gamma_i(t_i^{wcrt}) = \begin{cases} 1 \text{ si } lp_i^{NP}(t_i^{wcrt}) \neq \phi \\ 0 \text{ autrement} \end{cases} \quad \text{et } t_i^{NP} = \max_{\rho_k \in lp_i^{NP}(t_i^{wcrt})} (C_{sched}^{edf} + C_k + C_{term}^{edf} - 1)$$

Enfin, chaque charge C_{term}^{edf} prise en compte durant le calcul de $r_{i,t_i^{wcrt}}$ est obligatoirement précédée par l'exécution de la tâche associée ainsi que par une charge C_{sched}^{edf}. C'est pourquoi, le calcul de la durée totale d'exécution de la charge C_{sched}^{edf}, sur l'intervalle $[t_i^{wcrt}, t_i^{wcrt} + r_i[$ est conforme au calcul précédent à cela prêt que, pour chaque instance, nous déduisons une durée C_{term}^{edf} et la durée d'exécution pire cas de la tâche correspondante. De plus, si la tâche ρ_i est retardée par une tâche non préemptive moins prioritaire, nous devons considérer la charge C_{sched}^{edf} qui précède l'exécution de cette tâche moins prioritaire. Nous précisons que la complexité du calcul de r_i^{sched} est en $O(n^2.L)$.

$$r_i^{sched} = \sum_{\rho_j \in hp_i(t_i^{wcrt})} \sum_{t \in [0, t_i^{wcrt} + D_i - D_j] \cap [0, \overline{\omega}_i^{wcrt}]} \max(\min(r_{j,t} + t - C_{term}^{edf} - C_j - t_i^{wcrt}, C_{sched}^{edf}), 0) +$$
$$\sum_{t \in [s_i(t_i^{wcrt}), t_i^{wcrt}]} \max(\min(r_{i,t} + t - C_{term}^{edf} - C_i - t_i^{wcrt}, C_{sched}^{edf}), 0) +$$
$$\gamma_i(t_i^{wcrt}) \max(C_{sched}^{edf} - 1 - t_i^{wcrt}, 0)$$
$$\text{où } \gamma_i(t_i^{wcrt}) = \begin{cases} 1 \text{ si } lp_i^{NP}(t_i^{wcrt}) \neq \phi \\ 0 \text{ autrement} \end{cases}$$

5.f Synthèse

Ce chapitre nous a permis d'expliquer comment mettre en oeuvre un ordonnancement EDF, basé sur des priorités dynamiques, sur un noyau OSEK bien que celui-ci n'autorise que des tâches à priorité fixe. Les calculs des temps de réponse pires cas pour des tâches préemptives et non préemptives, tenant compte des charges cumulées de notre algorithme et du noyau OSEK, sont également détaillés. Une expérimentation de ces résultats, menée sur un ensemble de tâches périodiques ordonnancées selon la politique EDF, est présentée au chapitre 6. Cette expérimentation met en évidence l'intérêt de la politique EDF par rapport à la politique FP/FIFO et montre la validité de nos résultats pour le dimensionnement temps réel d'applications basées sur la politique EDF.

Chapitre 6

Expérimentations

Aux chapitres 4 et 5, nous avons amélioré les conditions de faisabilité classiques, respectivement pour les ordonnancements FP/FIFO et EDF, en intégrant, dans les équations nécessaires au calcul du temps de réponse pire cas de toute tâche périodique, les charges dues au système d'exploitation. Pour mémoire, ces charges résultent de la gestion des tâches, de la base de temps, des alarmes et du mécanisme du plafond de priorité qui protège l'accès à chaque ressource. Les conditions de faisabilité ainsi mises au point vont maintenant être expérimentées sur une plateforme réelle. A la section 6.a, nous expérimentons les conditions de faisabilité de l'ordonnancement FP/FIFO à l'aide d'un ensemble de dix tâches périodiques mis en oeuvre avec différentes valeurs de T_{tick}. Puis, une expérimentation est menée sur l'algorithme EDF implémenté selon les explications données à la section 5.a. Ainsi, la section 6.b montre les résultats de cette expérimentation et met en avant l'optimalité de la politique EDF à travers un ensemble de tâches périodiques pour lequel la méthode d'Audsley, décrite au paragraphe 2.b.i.3, ne trouve aucune solution. Enfin, les résultats précédents sont résumés à la section 6.c.

6.a Ordonnancement FP/FIFO mixte préemptif et non préemptif

Dans cette section, nous exposons nos résultats expérimentaux obtenus sur l'ensemble de tâches périodiques τ décrit au tableau 6.1 et ordonnancé selon la politique FP/FIFO. Les temps de réponse pires cas théoriques, tenant compte de l'exécution du système d'exploitation, découlent des conditions de faisabilité étudiées à la section 4.c. Les temps de réponse pires cas théoriques, ignorant les charges dues au noyau, sont calculés à partir des mêmes équations en annulant les six charges évoquées à la section 3.e. Enfin, les temps de réponse pires cas réels sont mesurés sur notre plateforme en prenant soin de reproduire, autant que possible, les scénarios pires cas correspondant. A titre indicatif, nous précisons, pour chaque temps de réponse pire cas théorique considérant les charges du noyau, la durée consacrée à l'exécution de chaque charge sur ce temps de réponse.

Tâche	P_i	C_i (μs)	T_i (μs)	D_i (μs)	B_i (μs)	*Ordonnancement*
τ_{10}	6	100	1000	50000	0	Non préemptive
τ_9	5	400	10000	40000	0	Non préemptive
τ_8	4	1000	45000	40000	500	Préemptive
τ_7	3	500	50000	50000	0	Non préemptive
τ_6	3	300	7500	50000	0	Non préemptive
τ_5	3	1000	15000	40000	500	Préemptive
τ_4	2	20000	300000	60000	0	Non préemptive
τ_3	1	3000	40000	80000	0	Préemptive
τ_2	1	10000	100000	80000	10000	Préemptive
τ_1	0	20000	180000	180000	0	Préemptive

TABLE 6.1 – Ensemble de dix tâches expérimenté avec l'ordonnancement FP/FIFO

Pour chaque tâche τ_i appartenant à τ, et pour chaque valeur prise par T_{tick}, nous nous intéressons aux durées suivantes :

- $r_{i,rel}$: Le temps de réponse pire cas réel mesuré sur la plateforme OSEK.
- $r_{i,th}^{aov}$: Le temps de réponse pire cas théorique prenant en compte les charges du noyau.
- $r_{i,th}^{sov}$: Le temps de réponse pire cas théorique ignorant les charges du noyau.
- r_i^{tick} : La durée d'exécution de la charge C_{tick} sur le temps de réponse pire cas $r_{i,th}^{aov}$.
- r_i^{act} : La durée d'exécution de la charge C_{act} sur le temps de réponse pire cas $r_{i,th}^{aov}$.
- r_i^{sched} : La durée d'exécution de la charge C_{sched} sur le temps de réponse pire cas $r_{i,th}^{aov}$.
- r_i^{term} : La durée d'exécution de la charge C_{term} sur le temps de réponse pire cas $r_{i,th}^{aov}$.
- r_i^{get} : La durée d'exécution de la charge C_{get} sur le temps de réponse pire cas $r_{i,th}^{aov}$.
- r_i^{rel} : La durée d'exécution de la charge C_{rel} sur le temps de réponse pire cas $r_{i,th}^{aov}$.

Les figures 6.1 à 6.20 montrent les résultats obtenus pour chaque tâche τ_i de l'ensemble τ quelque soit la valeur de T_{tick} prise dans l'ensemble $\Omega = \{(1+k) \times 100\mu s, k \in [0,9] \cap N\}$. Chacune de ces figures représente certaines des fonctions définies ci-dessous :

- $f = 100 \times (r_{i,th}^{aov} - r_{i,rel})/r_{i,rel}$: Déviation entre le temps de réponse pire cas théorique tenant compte du noyau et le temps de réponse pire cas réellement mesuré.

- $g = 100 \times (r_{i,th}^{sov} - r_{i,rel})/r_{i,rel}$: Déviation entre le temps de réponse pire cas théorique ignorant les charges du noyau et le temps de réponse pire cas réellement mesuré.

- $h = 100 \times (D_i - r_{i,th}^{aov})/r_{i,th}^{aov}$: Déviation entre l'échéance D_i de la tâche τ_i et son temps de réponse pire cas théorique tenant compte du noyau.

- $\alpha_{tick} = 100 \times r_i^{tick}/r_{i,th}^{aov}$: Pourcentage du temps consacré à la charge C_{tick} sur le temps de réponse pire cas théorique tenant compte du noyau.

- $\alpha_{act} = 100 \times r_i^{act}/r_{i,th}^{aov}$: Pourcentage du temps consacré à la charge C_{act} sur le temps de réponse pire cas théorique tenant compte du noyau.

- $\alpha_{sched} = 100 \times r_i^{sched}/r_{i,th}^{aov}$: Pourcentage du temps consacré à la charge C_{sched} sur le temps de réponse pire cas théorique tenant compte du noyau.

- $\alpha_{term} = 100 \times r_i^{term}/r_{i,th}^{aov}$: Pourcentage du temps consacré à la charge C_{term} sur le temps de réponse pire cas théorique tenant compte du noyau.

- $\alpha_{get} = 100 \times r_i^{get}/r_{i,th}^{aov}$: Pourcentage du temps consacré à la charge C_{get} sur le temps de réponse pire cas théorique tenant compte du noyau.

- $\alpha_{rel} = 100 \times r_i^{rel}/r_{i,th}^{aov}$: Pourcentage du temps consacré à la charge C_{rel} sur le temps de réponse pire cas théorique tenant compte du noyau.

Au regard du tableau 6.1, nous constatons que l'ensemble τ comporte des tâches préemptives et d'autres non préemptives. De plus, certaines tâches possèdent la même priorité. En conséquence, nous récapitulons les durées associées aux charges du noyau pour l'ensemble τ, au tableau 6.2, conformément aux fonctions de caractérisation établies à la section 3.e.iii.2.

C_{tick} $(cycles)$	C_{act} $(cycles)$	C_{sched} $(cycles)$	C_{term} $(cycles)$	C_{get} $(cycles)$	C_{rel} $(cycles)$
163	546	139	364	183	395

TABLE 6.2 – Ensemble de tâches expérimenté

La figure 6.1 montre les déviations f, g et h observées sur la tâche τ_1 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations précédentes vérifient respectivement $f \in [0.57\%, 11.41\%]$, $g \in [-24.31\%, -1.70\%]$ et $h \in [54.29\%, 104.83\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_1 respecte toujours son échéance temporelle. De plus, lorsque les charges du noyau sont ignorées, la théorie mène à un temps de réponse pire cas inférieur à la réalité.

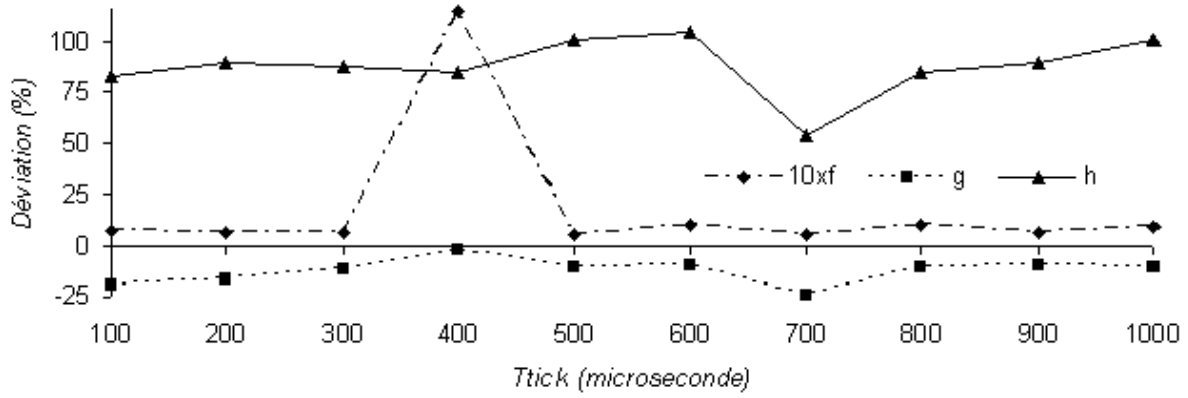

FIGURE 6.1 – Résultats de la tâche τ_1 : Déviations f, g et h

La figure 6.2 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_1 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.55\%, 5.53\%]$, $\alpha_{act} \in [2.36\%, 3.43\%]$, $\alpha_{sched} \in [0.60\%, 0.87\%]$, $\alpha_{term} \in [1.57\%, 2.29\%]$, $\alpha_{get} \in [0.06\%, 0.07\%]$ et $\alpha_{rel} \in [0.13\%, 0.16\%]$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} < \alpha_{rel}$.

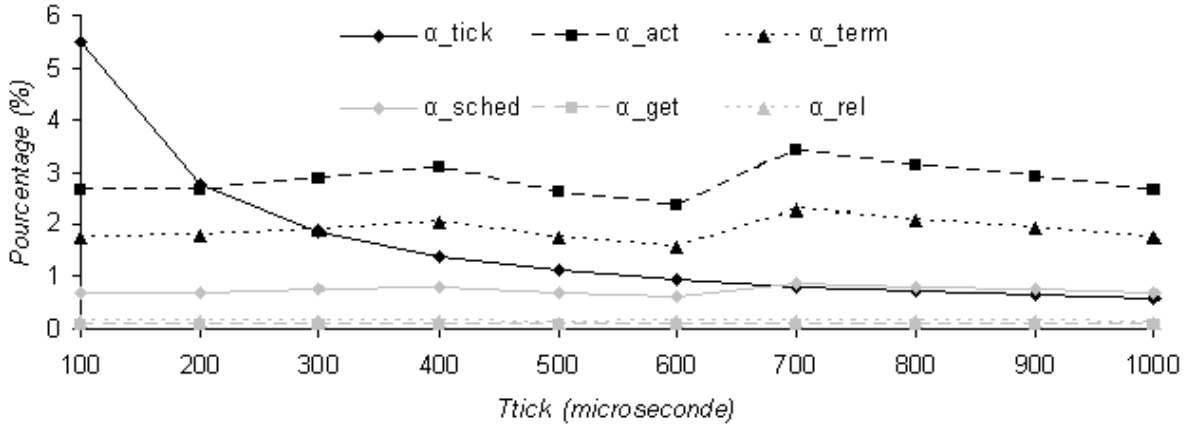

FIGURE 6.2 – Résultats de la tâche τ_1 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.3 montre les déviations f, g et h observées sur la tâche τ_2 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations précédentes vérifient respectivement $f \in [1.18\%, 9.42\%]$, $g \in [-13.41\%, -2.96\%]$ et $h \in [38.94\%, 52.91\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_2 respecte toujours son échéance temporelle. De plus, lorsque les charges du noyau sont ignorées, la théorie mène à un temps de réponse pire cas inférieur à la réalité.

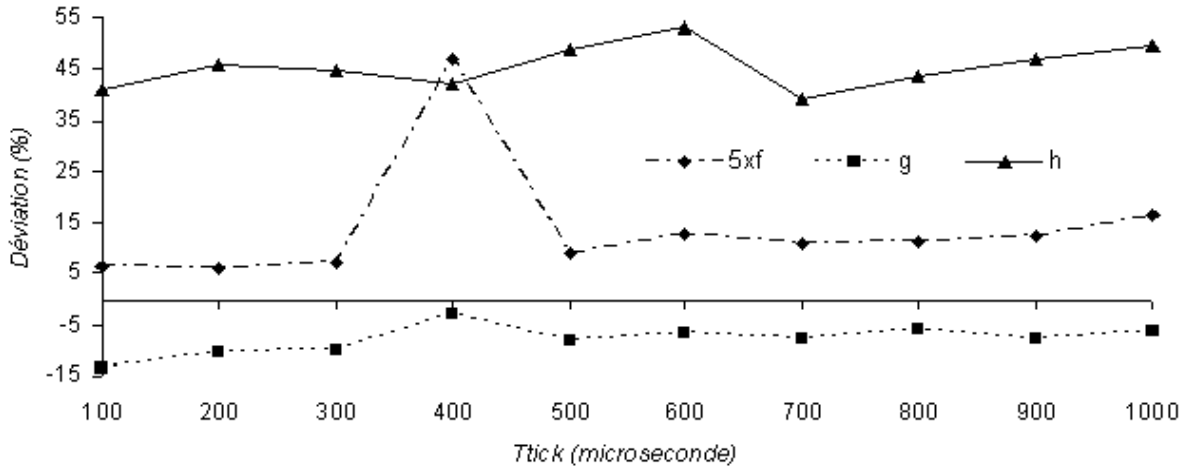

FIGURE 6.3 – Résultats de la tâche τ_2 : Déviations f, g et h

La figure 6.4 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_2 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.56\%, 5.53\%]$, $\alpha_{act} \in [2.51\%, 3.54\%]$, $\alpha_{sched} \in [0.62\%, 0.88\%]$, $\alpha_{term} \in [1.63\%, 2.32\%]$, $\alpha_{get} = 0.08\%$ et $\alpha_{rel} \in [0.16\%, 0.18\%]$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} < \alpha_{rel}$.

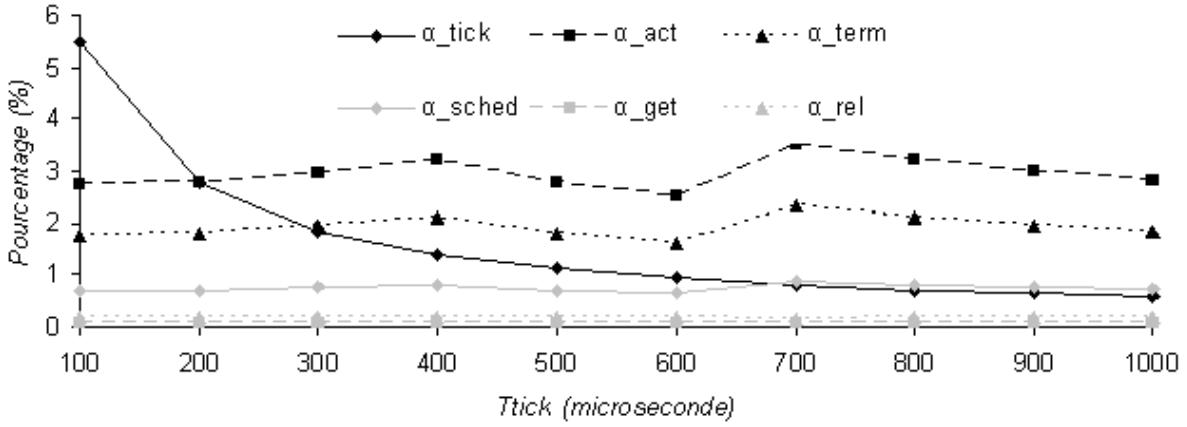

FIGURE 6.4 – Résultats de la tâche τ_2 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.5 montre les déviations f, g et h observées sur la tâche τ_3 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations précédentes vérifient respectivement $f \in [1.02\%, 8.85\%]$, $g \in [-13.44\%, -3.47\%]$ et $h \in [38.94\%, 52.91\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_3 respecte toujours son échéance temporelle. De plus, lorsque les charges du noyau sont ignorées, la théorie mène à un temps de réponse pire cas inférieur à la réalité.

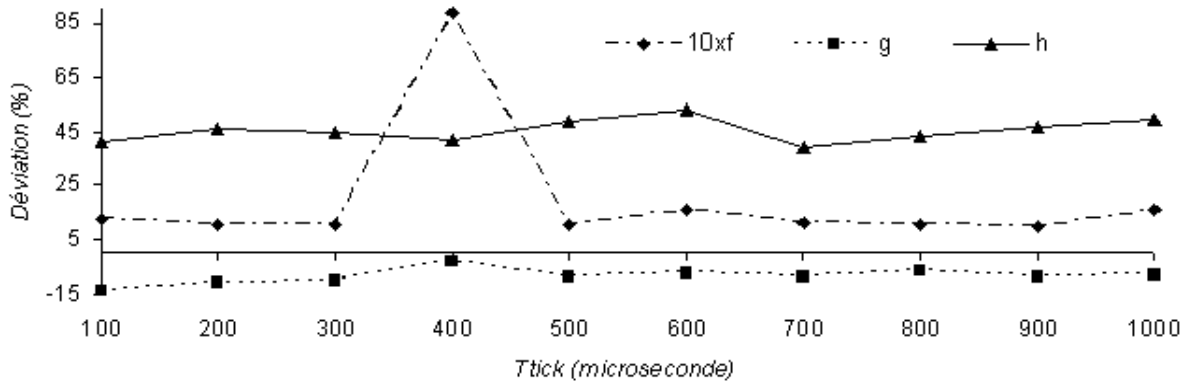

FIGURE 6.5 – Résultats de la tâche τ_3 : Déviations f, g et h

La figure 6.6 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_3 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.56\%, 5.53\%]$, $\alpha_{act} \in [2.51\%, 3.54\%]$, $\alpha_{sched} \in [0.62\%, 0.88\%]$, $\alpha_{term} \in [1.63\%, 2.32\%]$, $\alpha_{get} = 0.08\%$ et $\alpha_{rel} \in [0.16\%, 0.18\%]$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} < \alpha_{rel}$.

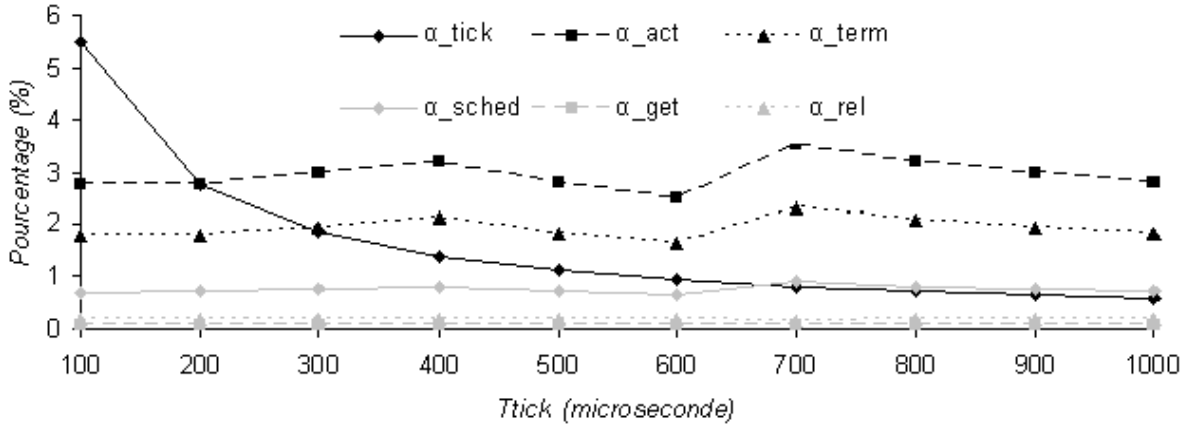

FIGURE 6.6 – Résultats de la tâche τ_3 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.7 montre les déviations f, g et h observées sur la tâche τ_4 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations précédentes vérifient respectivement $f \in [1.63\%, 6.11\%]$, $g \in [-6.28\%, -1.81\%]$ et $h \in [43.60\%, 55.81\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_4 respecte toujours son échéance temporelle. De plus, lorsque les charges du noyau sont ignorées, la théorie mène à un temps de réponse pire cas inférieur à la réalité.

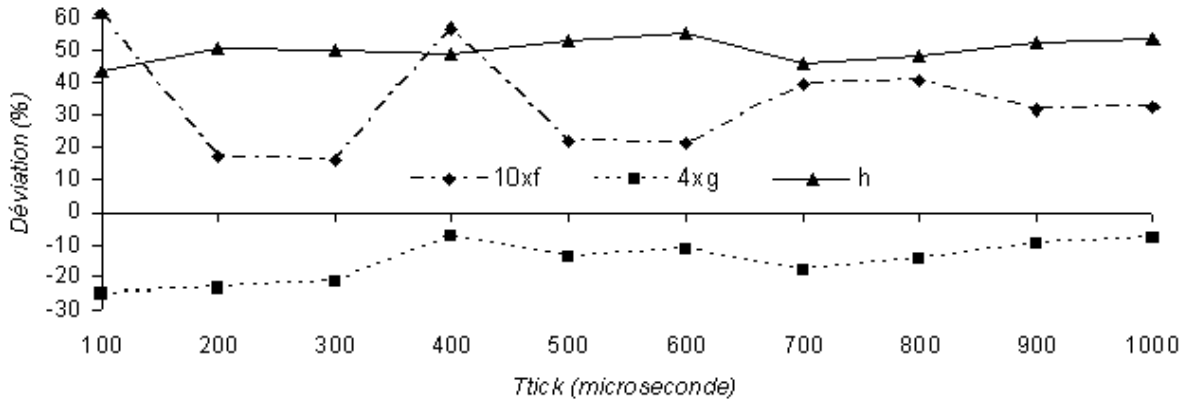

FIGURE 6.7 – Résultats de la tâche τ_4 : Déviations f, g et h

La figure 6.8 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_4 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.55\%, 5.53\%]$, $\alpha_{act} \in [2.50\%, 3.60\%]$, $\alpha_{sched} \in [0.31\%, 0.47\%]$, $\alpha_{term} \in [0.80\%, 1.23\%]$, $\alpha_{get} = 0.06\%$ et $\alpha_{rel} \in [0.13\%, 0.14\%]$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} < \alpha_{rel}$.

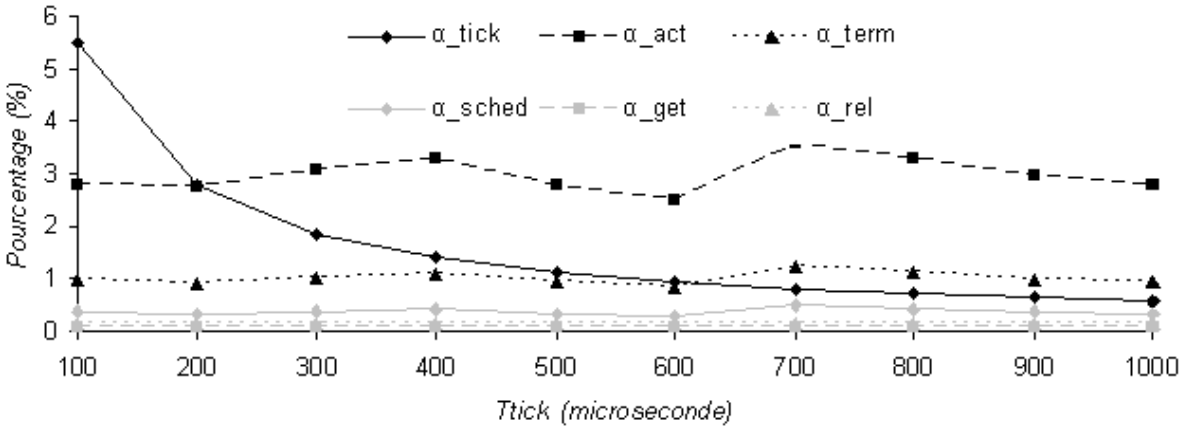

FIGURE 6.8 – Résultats de la tâche τ_4 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.9 montre les déviations f, g et h observées sur la tâche τ_5 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations f et h vérifient respectivement $f \in [1.84\%, 10.88\%]$ et $h \in [27.96\%, 43.11\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_5 respecte toujours son échéance temporelle. Concernant la déviation g, nous constatons que $\forall T_{tick} \in \Omega - \{900\mu s\}, g \in [-11.68\%, -0.43\%]$. Cependant, lorsque T_{tick} vaut $900\mu s$, la déviation g devient positive et atteint 0.67%. Ceci montre que le temps de réponse pire cas ignorant les charges du noyau est toujours inférieur à la réalité sauf lorsque T_{tick} vaut $900\mu s$ car, dans ce cas, nous n'avons pu reproduire le scénario pire cas.

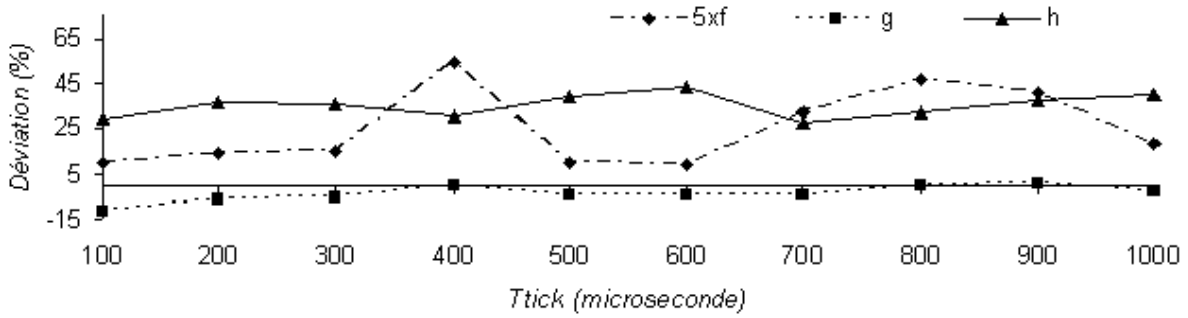

FIGURE 6.9 – Résultats de la tâche τ_5 : Déviations f, g et h

La figure 6.10 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_5 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.56\%, 5.54\%]$, $\alpha_{act} \in [2.58\%, 3.73\%]$, $\alpha_{sched} \in [0.52\%, 0.80\%]$, $\alpha_{term} \in [1.46\%, 2.17\%]$, $\alpha_{get} = 0.04\%$ et $\alpha_{rel} \in [0.09\%, 0.10\%]$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} < \alpha_{rel}$.

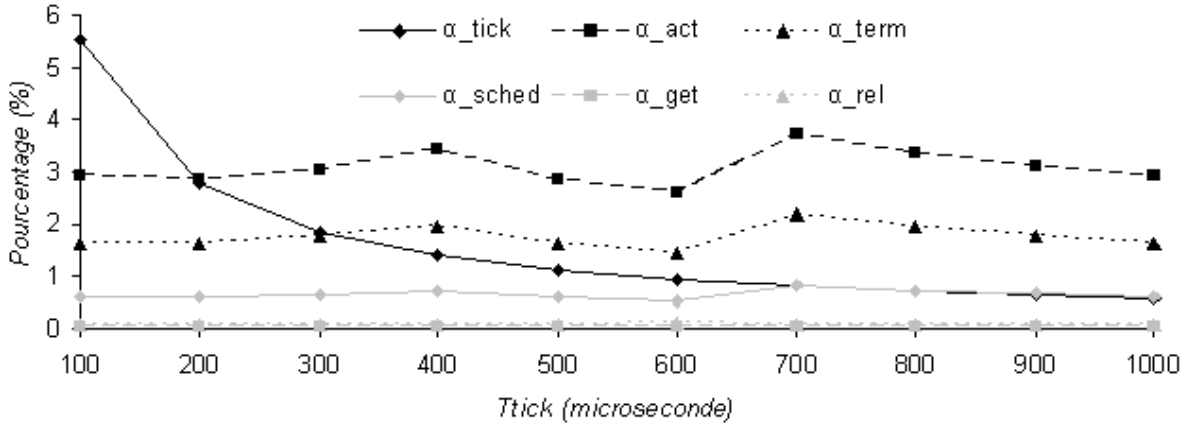

FIGURE 6.10 – Résultats de la tâche τ_5 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.11 montre les déviations f, g et h observées sur la tâche τ_6 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations f et h vérifient respectivement $f \in [2.06\%, 11.02\%]$ et $h \in [60.55\%, 79.64\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_6 respecte toujours son échéance temporelle. Concernant la déviation g, nous constatons que $\forall T_{tick} \in \Omega - \{400\mu s, 900\mu s, 1000\mu s\}, g \in [-9.92\%, -0.80\%]$. Cependant, lorsque T_{tick} vaut $400\mu s$, $900\mu s$ ou $1000\mu s$, la déviation g devient positive et peut atteindre 1.73%. Ceci montre que le temps de réponse pire cas ignorant les charges du noyau est toujours inférieur à la réalité sauf lorsque T_{tick} vaut $400\mu s$, $900\mu s$ ou $1000\mu s$ car, dans chacun de ces trois cas, nous n'avons pu reproduire le scénario pire cas.

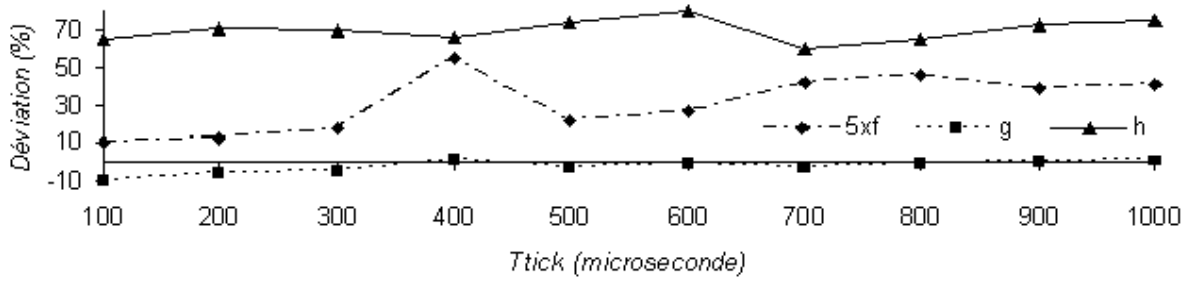

FIGURE 6.11 – Résultats de la tâche τ_6 : Déviations f, g et h

La figure 6.12 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_6 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.56\%, 5.54\%]$, $\alpha_{act} \in [2.59\%, 3.75\%]$, $\alpha_{sched} \in [0.51\%, 0.79\%]$, $\alpha_{term} \in [1.42\%, 2.14\%]$, $\alpha_{get} = 0.04\%$ et $\alpha_{rel} \in [0.09\%, 0.10\%]$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} < \alpha_{rel}$.

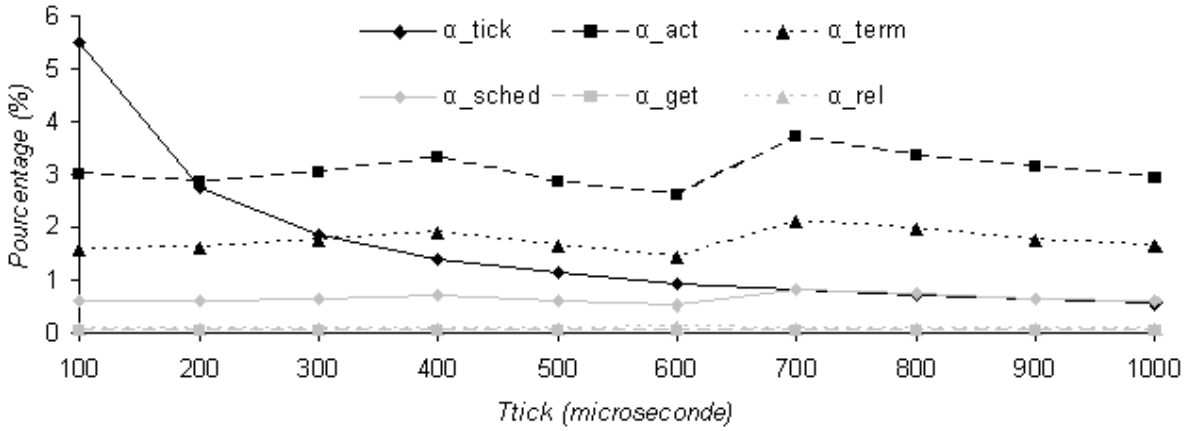

FIGURE 6.12 – Résultats de la tâche τ_6 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.13 montre les déviations f, g et h observées sur la tâche τ_7 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations f et h vérifient respectivement $f \in [2.16\%, 10.05\%]$ et $h \in [60.55\%, 79.64\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_7 respecte toujours son échéance temporelle. Concernant la déviation g, nous constatons que $\forall T_{tick} \in \Omega - \{400\mu s\}$, $g \in [-9.84\%, -1.36\%]$. Cependant, lorsque T_{tick} vaut $400\mu s$, la déviation g devient positive et atteint 0.83%. Ceci montre que le temps de réponse pire cas ignorant les charges du noyau est toujours inférieur à la réalité sauf lorsque T_{tick} vaut $400\mu s$ car, dans ce cas, nous n'avons pu reproduire le scénario pire cas.

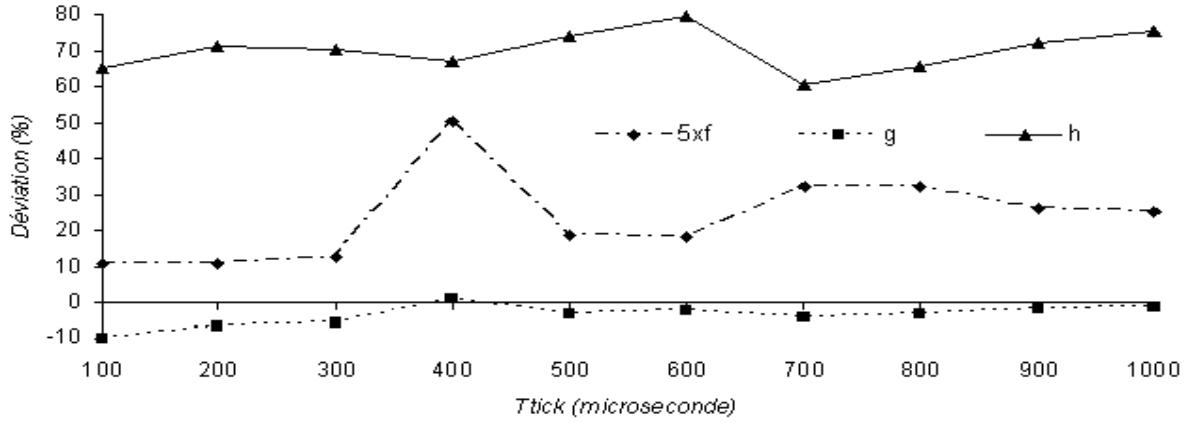

FIGURE 6.13 – Résultats de la tâche τ_7 : Déviations f, g et h

La figure 6.14 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_7 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.56\%, 5.54\%]$, $\alpha_{act} \in [2.59\%, 3.75\%]$, $\alpha_{sched} \in [0.51\%, 0.79\%]$, $\alpha_{term} \in [1.42\%, 2.14\%]$, $\alpha_{get} = 0.04\%$ et $\alpha_{rel} \in [0.09\%, 0.10\%]$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} < \alpha_{rel}$.

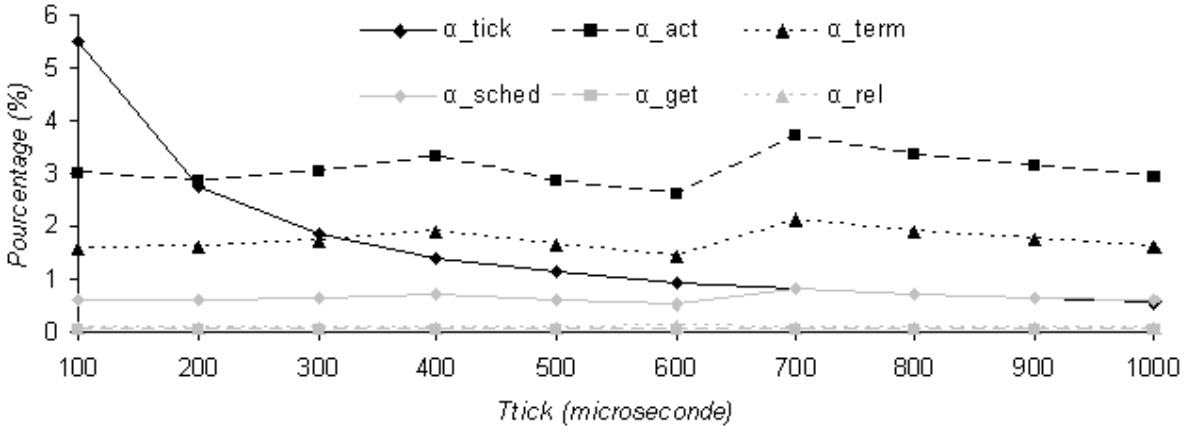

FIGURE 6.14 – Résultats de la tâche τ_7 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.15 montre les déviations f, g et h observées sur la tâche τ_8 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations f et h vérifient respectivement $f \in [1.57\%, 9.62\%]$ et $h \in [41.08\%, 55.12\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_8 respecte toujours son échéance temporelle. Concernant la déviation g, nous constatons que $\forall T_{tick} \in \Omega - \{400\mu s\}$, $g \in [-10.09\%, -1.66\%]$. Cependant, lorsque T_{tick} vaut $400\mu s$, la déviation g devient positive et atteint 0.52%.Ceci montre que le temps de réponse pire cas ignorant les charges du noyau est toujours inférieur à la réalité sauf lorsque T_{tick} vaut $400\mu s$ car, dans ce cas, nous n'avons pu reproduire le scénario pire cas.

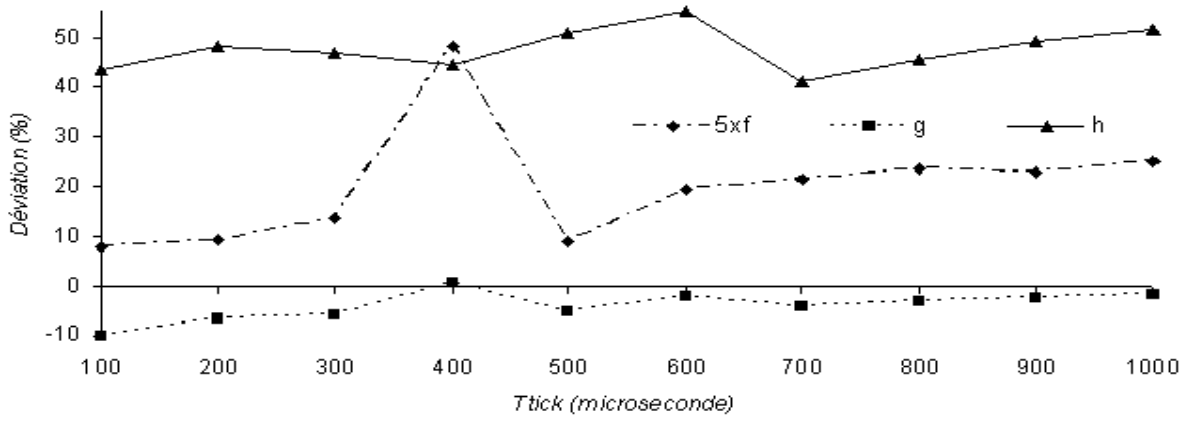

FIGURE 6.15 – Résultats de la tâche τ_8 : Déviations f, g et h

La figure 6.16 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_8 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.57\%, 5.53\%]$, $\alpha_{act} \in [2.66\%, 3.66\%]$, $\alpha_{sched} \in [0.48\%, 0.75\%]$, $\alpha_{term} \in [1.34\%, 2.05\%]$, $\alpha_{get} = 0.02\%$ et $\alpha_{rel} = 0.05\%$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} < \alpha_{rel}$.

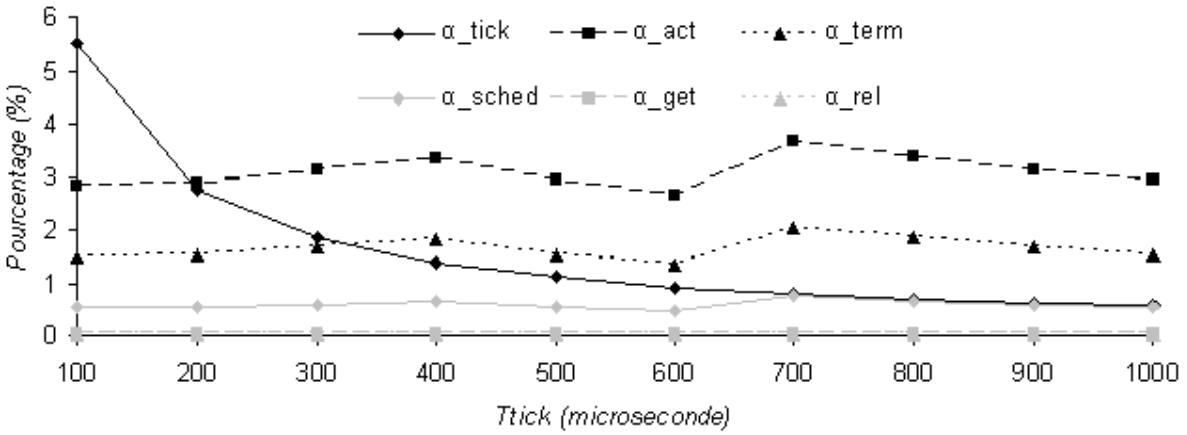

FIGURE 6.16 – Résultats de la tâche τ_8 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.17 montre les déviations f, g et h observées sur la tâche τ_9 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations f et h vérifient respectivement $f \in [1.97\%, 10.36\%]$ et $h \in [54.20\%, 69.29\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_9 respecte toujours son échéance temporelle. Concernant la déviation g, nous constatons que $\forall T_{tick} \in \Omega - \{400\mu s\}$, $g \in [-9.56\%, -0.66\%]$. Cependant, lorsque T_{tick} vaut $400\mu s$, la déviation g devient positive et atteint 1.29%.Ceci montre que le temps de réponse pire cas ignorant les charges du noyau est toujours inférieur à la réalité sauf lorsque T_{tick} vaut $400\mu s$ car, dans ce cas, nous n'avons pu reproduire le scénario pire cas.

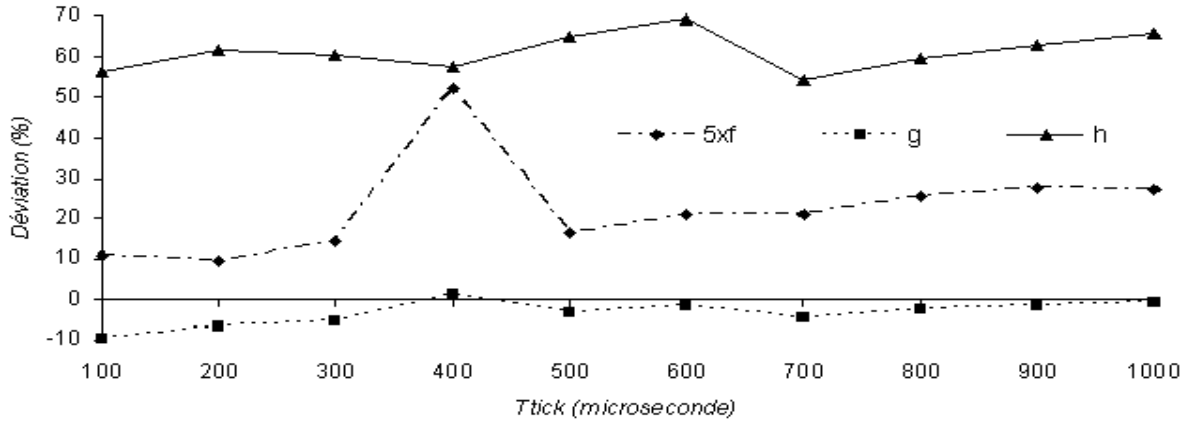

FIGURE 6.17 – Résultats de la tâche τ_9 : Déviations f, g et h

La figure 6.18 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_9 pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.57\%, 5.54\%]$, $\alpha_{act} \in [2.74\%, 3.78\%]$, $\alpha_{sched} \in [0.42\%, 0.69\%]$, $\alpha_{term} \in [1.20\%, 1.90\%]$, $\alpha_{get} = 0.00\%$ et $\alpha_{rel} = 0.00\%$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} = \alpha_{rel}$.

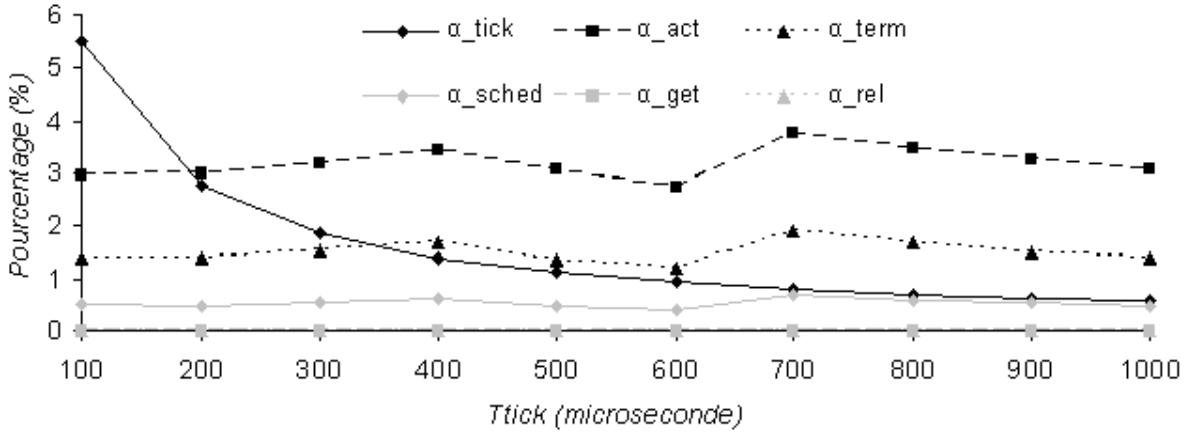

FIGURE 6.18 – Résultats de la tâche τ_9 : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

La figure 6.19 montre les déviations f, g et h observées sur la tâche τ_{10} pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les déviations f et h vérifient respectivement $f \in [0.95\%, 4.53\%]$ et $h \in [127.28\%, 139.45\%]$. Ceci signifie que le temps de réponse pire cas tenant compte du noyau est toujours supérieur au cas réel et que τ_{10} respecte toujours son échéance temporelle. Concernant la déviation g, nous constatons que $\forall T_{tick} \in \Omega - \{1000\mu s\}, g \in [-7.77\%, -0.53\%]$. Cependant, lorsque T_{tick} vaut $1000\mu s$, la déviation g devient positive et atteint 0.62%. Ceci montre que le temps de réponse pire cas ignorant les charges du noyau est toujours inférieur à la réalité sauf lorsque T_{tick} vaut $1000\mu s$ car, dans ce cas, nous n'avons pu reproduire le scénario pire cas.

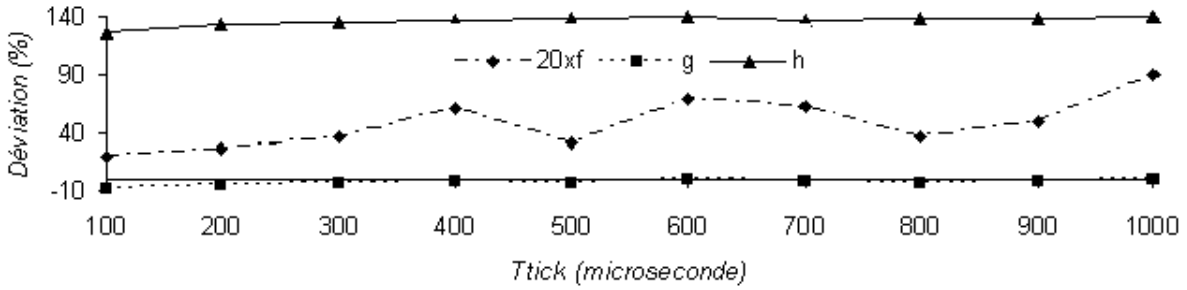

FIGURE 6.19 – Résultats de la tâche τ_{10} : Déviations f, g et h

La figure 6.20 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur la tâche τ_{10} pour chaque valeur de T_{tick} dans l'intervalle Ω. Nous observons que, $\forall T_{tick} \in \Omega$, les pourcentages précédents vérifient respectivement $\alpha_{tick} \in [0.56\%, 5.53\%]$, $\alpha_{act} \in [2.83\%, 3.94\%]$, $\alpha_{sched} = 0.02\%$, $\alpha_{term} \in [0.17\%, 0.18\%]$, $\alpha_{get} = 0.00\%$ et $\alpha_{rel} = 0.00\%$. Nous constatons également que α_{tick} est inversement proportionnel à la valeur T_{tick} et que, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act}$ et $\alpha_{get} = \alpha_{rel}$.

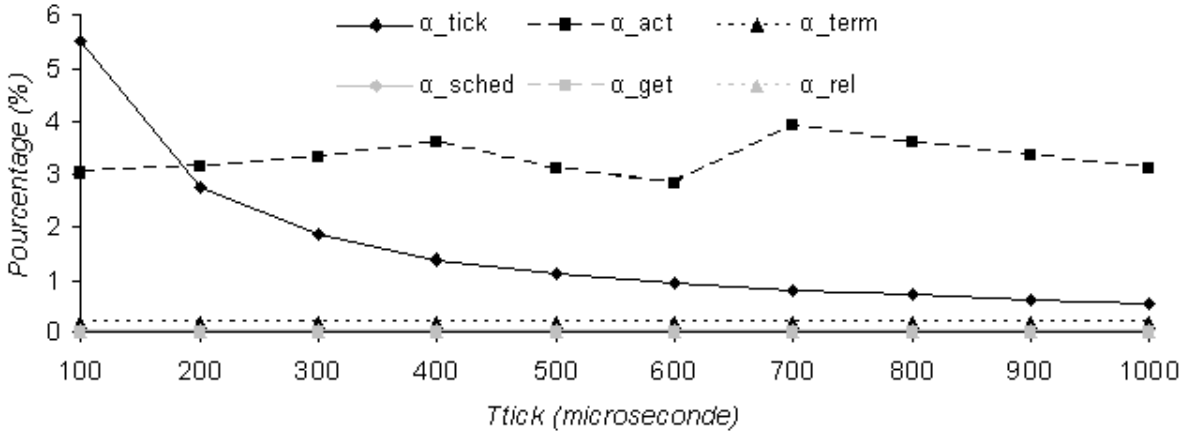

FIGURE 6.20 – Résultats de la tâche τ_{10} : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

Premièrement, nous constatons que les courbes, présentées aux figures 6.1 à 6.20, sont toutes discontinues hormis α_{tick}. Ceci s'explique par le fait que, pour chaque valeur de T_{tick} appartenant à Ω, la période de chaque tâche de l'ensemble τ décrit au tableau 6.1 est arrondie au multiple de T_{tick} le plus proche selon la formule donnée à la section 3.d. Ainsi, pour chaque valeur de T_{tick}, l'ensemble τ apparaît différent. La courbe α_{tick}, quant à elle, montre le pourcentage de temps consacré à la charge C_{tick} sur le temps de réponse pire cas théorique prenant en compte les charges du noyau. Comme expliqué à la sous section 3.e.i, le facteur d'utilisation du processeur associé à la charge C_{tick} vaut $U_{tick} = C_{tick}/T_{tick}$, ce qui justifie que le pourcentage α_{tick} apparaisse toujours inversement proportionnel à la valeur de T_{tick}.

Concernant le pourcentage α_{tick}, conformément à ce que nous venons d'expliquer au paragraphe ci-dessus, celui-ci croît d'un facteur 10 lorsque T_{tick} passe de $1000\mu s$ à $100\mu s$. Quant aux pourcentages, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}, nous remarquons, $\forall T_{tick} \in \Omega$, $\alpha_{sched} < \alpha_{term} < \alpha_{act} \leq 3.94\%$ et $\alpha_{get} \leq \alpha_{rel} \leq 0.18\%$. Nous précisons que, dans les inégalités précédentes, les bornes supérieures égales à 3.94% et 0.18% correspondent respectivement aux valeurs maximales de α_{act}, atteinte avec la tâche τ_{10} lorsque $T_{tick} = 700\mu s$, et de α_{rel}, atteinte avec les tâches τ_2 et τ_3 lorsque $T_{tick} = 600$ ou $1000\mu s$. Comme l'indiquent les formules r_i^{get} et r_i^{rel} fournies aux paragraphes 4.c.iv.2 et 4.c.v.2, lorsqu'une charge C_{rel} survient durant le temps de réponse d'une tâche, la charge C_{get} qui la précède peut s'exécuter avant l'activation de cette même tâche et ainsi ne pas être considérée dans son temps de réponse. A l'inverse, lorsqu'une charge C_{get} apparaît durant le temps de réponse d'une tâche, la charge C_{rel} qui lui succède se produit obligatoirement dans le même temps de réponse. Finalement, sur un temps de réponse donné, nous aurons toujours au moins autant de charges C_{rel} que de charges C_{get}. Sachant que $C_{get} < C_{rel}$, nous comprenons aisément que, $\forall T_{tick} \in \Omega$, $\alpha_{get} \leq \alpha_{rel}$. Par rapport à la tâche τ_{10}, d'après la figure 6.20, les pourcentages α_{get} et α_{rel} sont toujours nuls. Ceci s'explique par le fait que cette tâche n'accède pas à la ressource et que, dans son scénario pire cas, celle-ci n'est pas retardée à cause du mécanisme du plafond de priorité mais par la tâche non préemptive τ_4. Sachant que la tâche non préemptive τ_4 retarde la tâche τ_{10}, celle-ci débute son exécution une fois le service *TerminateTask* appelé par la tâche τ_4. Le service *TerminateTask* va alors élire la tâche τ_{10} et accomplir le changement de contexte. De même, au niveau de la tâche τ_9, les pourcentages α_{get} et α_{rel} sont nuls car celle-ci, comme la tâche τ_{10}, ne recourt pas à la ressource et n'est pas retardée par le mécanisme du plafond de priorité.

Quant aux déviations f, g et h, nous observons que, $\forall T_{tick} \in \Omega$ et $\forall \tau_i \in \tau$, $f \in [0.57\%, 11.41\%]$, $g \in [-24.31\%, 1.73\%]$ et $h \in [27.96\%, 139.45\%]$. Le fait que la déviation f soit toujours strictement positive montre que le temps de réponse pire cas théorique tenant compte des charges du noyau, calculé conformément aux équations fournies aux sous sections 4.c.iv et 4.c.v, est toujours strictement supérieur au temps de réponse réel. Ces équations répondent donc à notre attente et s'avèrent utiles au dimensionnement temps réel d'applications basées sur un noyau OSEK. De plus, leur surestimation n'excédant pas 11.41%, celles-ci se révèlent plus économiques que l'approche courante des développeurs qui consiste à réserver 30% de la charge du processeur au système d'exploitation.

La positivité de la déviation h indique que, selon les temps de réponse pire cas théoriques calculés en tenant compte des charges noyau, chaque tâche de l'ensemble τ respecte son échéance temporelle. Comme nous l'avons constaté, ces temps de réponse pires cas théoriques sont toujours supérieurs aux temps de réponse réels, ainsi la déviation h permet bien de valider l'ensemble τ. Enfin, nous précisons que la déviation g n'est positive qu'en huit points : avec la tâche τ_5 lorsque T_{tick} vaut $900\mu s$, avec la tâche τ_6 lorsque T_{tick} vaut $400\mu s$, $900\mu s$ ou $1000\mu s$, avec les tâches τ_7, τ_8 et τ_9 lorsque T_{tick} vaut $400\mu s$ et avec la tâche τ_{10} lorsque T_{tick} vaut $1000\mu s$. La raison pour laquelle la déviation g est positive en ces points réside en notre incapacité à y reproduire le scénario pire cas. Le temps de réponse pire cas théorique ignorant les charges du noyau peut alors dépasser le temps de réponse réel. En tout autre point, la déviation g est strictement négative et appartient à l'intervalle $[-24.31\%, -0.43\%]$. La négativité de cette déviation montre que le temps de réponse pire cas théorique est inférieur à la réalité lorsque celui-ci ne tient pas compte des charges du noyau. Or, la déviation g est strictement négative dans 92% des tests effectués ce qui confirme l'importance de la considération de ces charges dans les conditions de faisabilité.

6.b Ordonnancement EDF mixte préemptif et non préemptif

Dans cette section, nous expérimentons l'ensemble de tâches périodiques ρ décrit au tableau 6.3 avec deux politiques d'ordonnancement différentes. A la sous section 6.b.i, nous présentons les résultats obtenus avec la politique FP/FIFO prescrite par le standard OSEK. Puis, l'algorithme EDF étant mis en oeuvre suivant la méthode exposée à la section 5.a, la sous section 6.b.ii relate l'expérimentation menée avec cette politique d'ordonnancement. Dans les deux cas, nous fixons le paramètre T_{tick} à $200\mu s$ et comparons les temps de réponse pires cas théoriques au temps de réponse réels mesurés sur notre plateforme en reproduisant les scénarios pires cas. Pour chaque temps de réponse théorique tenant compte des charges dues au noyau, nous précisons la durée consacrée à l'exécution de chacune de ces charges. Nous soulignons, qu'en l'absence de ressource partagée, aucune tâche ρ_i ne possède de paramètre B_i.

Tâche	C_i (μs)	T_i (μs)	D_i (μs)	*Ordonnancement*
ρ_5	600	6400	3000	Non préemptive
ρ_4	3000	14800	8000	Préemptive
ρ_3	2000	10800	10800	Non préemptive
ρ_2	1500	12000	6000	Non préemptive
ρ_1	10000	100000	33000	Préemptive

TABLE 6.3 – Ensemble de cinq tâches expérimenté avec les ordonnancements FP/FIFO et EDF

6.b.i Expérimentation avec la politique d'ordonnancement FP/FIFO

L'algorithme d'attribution de priorités fixes d'Audsley, présenté au paragraphe 2.b.i.3, est optimal pour des ensembles de tâches périodiques préemptives ou non préemptives. Autrement dit, si une solution, à base de priorités fixes, existe, alors celle-ci sera obligatoirement trouvée par cet algorithme. Lorsqu'appliquée à l'ensemble ρ décrit au tableau 6.3, en se basant sur nos conditions de faisabilité tenant compte des charges dues au système d'exploitation, cette méthode montre qu'aucune solution à base de priorités fixes n'existe pour celui-ci. Ainsi, nous avons arbitrairement fixé les priorités comme indiquées au tableau 6.4.

Tâche	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5
Priorité	0	1	2	3	4

TABLE 6.4 – Priorités fixes attibuées aux tâches décrites au tableau 6.3

Conformément à leur définition à la section 6.a, le tableau 6.5 récapitule les durées mesurées et calculées pour l'ensemble de tâches ρ avec un paramètre T_{tick} égal à $200\mu s$ ainsi que leurs échéances. Concernant les charges C_{tick}, C_{act}, C_{sched} et C_{term}, les valeurs utilisées suivent les fonctions de caractérisation établies à la sous section 3.e.iii.2 et figurent au tableau 6.6.

Tâche	*Échéance*	$r_{i,rel}$	$r_{i,th}^{aov}$	$r_{i,th}^{sov}$	r_i^{tick}	r_i^{act}	r_i^{sched}	r_i^{term}
ρ_5	88474	79065	81863	76678	2282	2400	139	1091
ρ_4	235930	166210	173448	165152	4890	2400	278	1455
ρ_3	318505	211738	219972	209389	6194	2880	417	1819
ρ_2	176948	217999	239162	209389	6683	2880	695	1820
ρ_1	973210	1077005	1128374	869995	31296	8640	2502	6552

TABLE 6.5 – Echéances et durées, exprimées en cycles, pour l'ensemble ρ ordonnancé avec FP/FIFO

C_{tick} $(cycles)$	C_{act} $(cycles)$	C_{sched} $(cycles)$	C_{term} $(cycles)$
163	480	139	364

TABLE 6.6 – Charges du noyau pour l'ordonnancement FP/FIFO de l'ensemble ρ

La figure 6.21 montre les déviations f, g et h définies à la section 6.a. Nous observons que, $\forall \rho_i \in \rho$, les déviations f et g vérifient respectivement $f \in [3.54\%, 9.71\%]$ et $g \in [-19.22\%, -0.64\%]$. Ainsi, selon que les charges du noyau soient prises en compte ou ignorées, le temps de réponse pire cas théorique est respectivement supérieur ou inférieur au temps de réponse réel. Concernant la déviation h, nous constatons que celle-ci vaut respectivement -13.75% et -26.01% pour les tâches ρ_1 et ρ_2 qui, effectivement, ne respectent pas leur échéance. Autrement, les trois autres tâches respectent leur échéance : $\forall \rho_j \in \rho - \{\rho_1, \rho_2\}, h \in [8.08\%, 44.79\%]$. Nous soulignons que, d'après les temps de réponse théoriques calculés en ignorant les charges du noyau, la tâche ρ_1 devait respecter son échéance. Ce constat met, à nouveau, en avant l'importance de la prise en compte de ces charges.

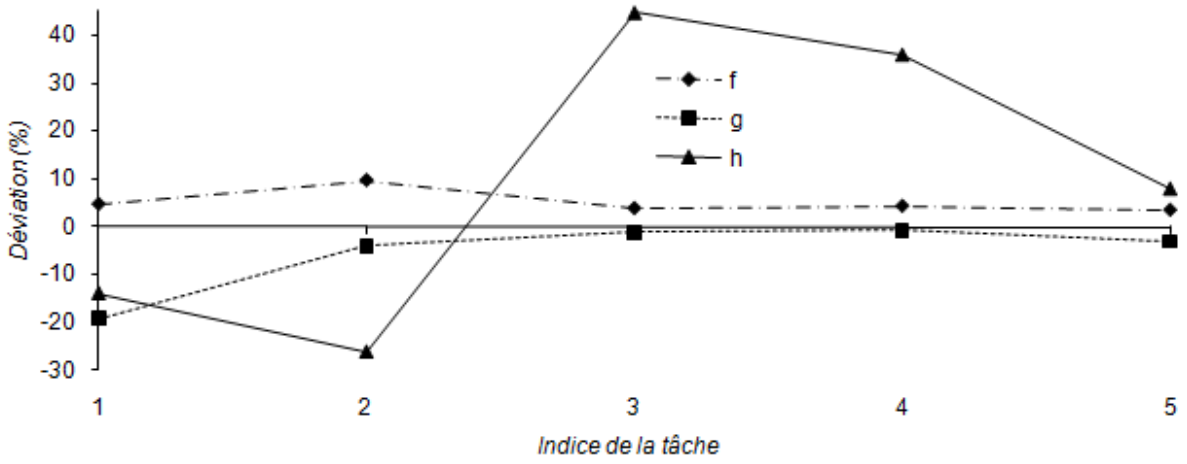

FIGURE 6.21 – Résultats avec FP/FIFO : Déviations f, g et h

La figure 6.22 montre les pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel} observés sur chaque tâche de l'ensemble ρ. Tout comme lors de l'expérimentation dont les résultats sont détaillés à la section 6.a, nous observons que $\forall \rho_i \in \rho, \alpha_{sched} < \alpha_{term} < \alpha_{act} \leq 2.93\%$. Nous précisons que, dans le cas de cette expérimentation, un cycle dure $33.9ns$. Ainsi, les temps de réponse pire cas théoriques, tenant compte des charges du noyau, des tâches ρ_4 et ρ_5 s'avèrent strictement inférieurs à l'ensemble des périodes. C'est pourquoi, pour chacune de ces deux tâches, seules les cinq activations survenant à la date 0 interviennent dans leur temps de réponse pire cas, soit $5 \times 480 = 2400$ *cycles*. L'importance du pourcentage α_{act}, au niveau de la tâche ρ_5, tient au fait que celle-ci possède le temps de réponse pire cas théorique le plus faible. Comme discuté à la section 6.a, le pourcentage α_{tick} dépend essentiellement de la période T_{tick}. Cette période valant toujours $200\mu s$ dans cette expérimentation, le pourcentage α_{tick} apparaît quasiment constant et vérifie $\forall \rho_i \in \rho, \alpha_{tick} \in [2.77\%, 2.82\%]$.

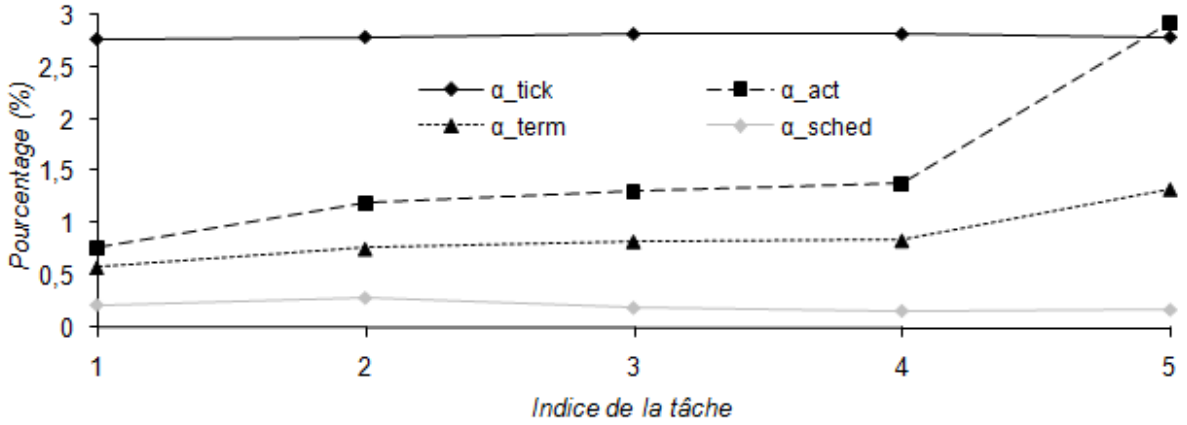

FIGURE 6.22 – Résultats avec FP/FIFO : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

6.b.ii Expérimentation avec la politique d'ordonnancement EDF

Face à l'échec de son ordonnancement avec la politique FP/FIFO, nous expérimentons maintenant l'ensemble de tâches ρ décrit au tableau 6.3 avec l'algorithme EDF. Nous conservons une période T_{tick} égale à $200\mu s$. Les charges cumulées du noyau et de l'algorithme EDF sont précisées au tableau 6.7. Les échéances ainsi que les durées mesurées et calculées pour l'ensemble de tâches ρ sont ensuite récapitulées au tableau 6.8.

$C_{tick}^{edf}\ (cycles)$	$C_{act}^{edf}\ (cycles)$	$C_{sched}^{edf}\ (cycles)$	$C_{term}^{edf}\ (cycles)$
538	68	845	268

TABLE 6.7 – Charges du noyau pour l'ordonnancement FP/FIFO de l'ensemble ρ

Tâche	*Échéance*	$r_{i,rel}$	$r_{i,th}^{aov}$	$r_{i,th}^{sov}$	r_i^{tick}	r_i^{act}	r_i^{sched}	r_i^{term}
ρ_5	88474	83575	87113	76678	8070	68	845	536
ρ_4	235930	225050	234656	209389	21520	408	4224	1340
ρ_3	318505	233771	256729	209389	23672	408	5070	1608
ρ_2	176948	166234	175666	150399	16140	136	3380	1072
ρ_1	973210	957998	966041	869995	89308	1156	12675	4020

TABLE 6.8 – Echéances et durées, exprimées en cycles, pour l'ensemble ρ ordonnancé avec FP/FIFO

La figure 6.23 montre les déviations f, g et h définies à la section 6.a. Nous observons que, pour toute tâche de l'ensemble ρ, les déviations f, g et h vérifient respectivement $f \in [0.84\%, 9.82\%]$, $g \in [-10.43\%, -6.96\%]$ et $h \in [0.54\%, 24.06\%]$. Ceci signifie que les temps de réponse pires cas théoriques, calculés en tenant compte des charges cumulées du noyau et de l'algorithme EDF, sont toujours supérieurs aux temps de réponse réels. Au contraire, les temps de réponse pires cas théoriques classiques apparaissent nettement inférieurs aux temps de réponse réels. La pointe observée sur la déviation f, au niveau de la tâche ρ_3, tient au fait que nous n'avons pu reproduire fidèlement le scénario pire cas correspondant à cette tâche. Par ailleurs, la déviation h étant toujours strictement positive, nous précisons que toutes les échéances sont honorées. Bien que l'ensemble ρ ne soit pas ordonnançable avec la politique FP/FIFO, l'algorithme EDF trouve donc une solution pour celui-ci.

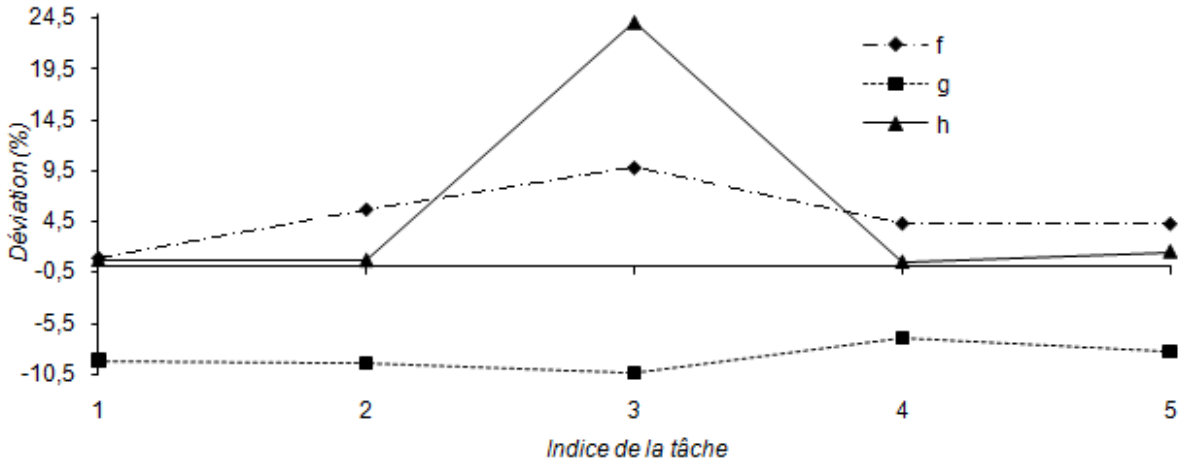

FIGURE 6.23 – Résultats avec EDF : Déviations f, g et h

La figure 6.24 montre les pourcentages α_{tick}, α_{act}, α_{sched} et α_{term} observés sur chaque tâche de l'ensemble ρ ordonnancé selon la politique EDF. Nous observons que, quelque soit la tâche de l'ensemble ρ, $\alpha_{act} < \alpha_{term} < \alpha_{sched} \leq 1.97\%$. Bien que nous ayons $C^{edf}_{act} < C^{edf}_{term} < C^{edf}_{sched}$, nous pourrions avoir un pourcentage α_{act} supérieur ou égal à α_{term} ainsi qu'à α_{sched}. En effet, lors du calcul du temps de réponse d'une tâche, préemptive ou non, nous considérons l'ensemble des activations qui surviennent sans intégrer toutes les exécutions associées. Par ailleurs, supposons que nous calculions le temps de réponse pire cas d'une tâche ρ_i. Lors de l'exécution d'une tâche ρ_j, la charge C^{edf}_{sched} précède l'exécution de la tâche et la charge C^{edf}_{term}. Ainsi, la charge C^{edf}_{sched} peut intervenir avant que la tâche ρ_i n'ait été activée alors que la charge C^{edf}_{term} intervient après cette activation. Nous considérons donc toujours, durant le calcul d'un temps de réponse pire cas, au moins autant de charges C^{edf}_{term} que de charges C^{edf}_{sched}. Nous pourrions donc avoir un pourcentage α_{term} supérieur ou égal à α_{sched}. Comme toujours, l'impact de la charge C^{edf}_{tick} dépendant essentiellement de la période T_{tick}, le pourcentage α_{tick} apparaît quasiment constant et vérifie $\forall \rho_i \in \rho, \alpha_{tick} \in [9.17\%, 9.26\%]$.

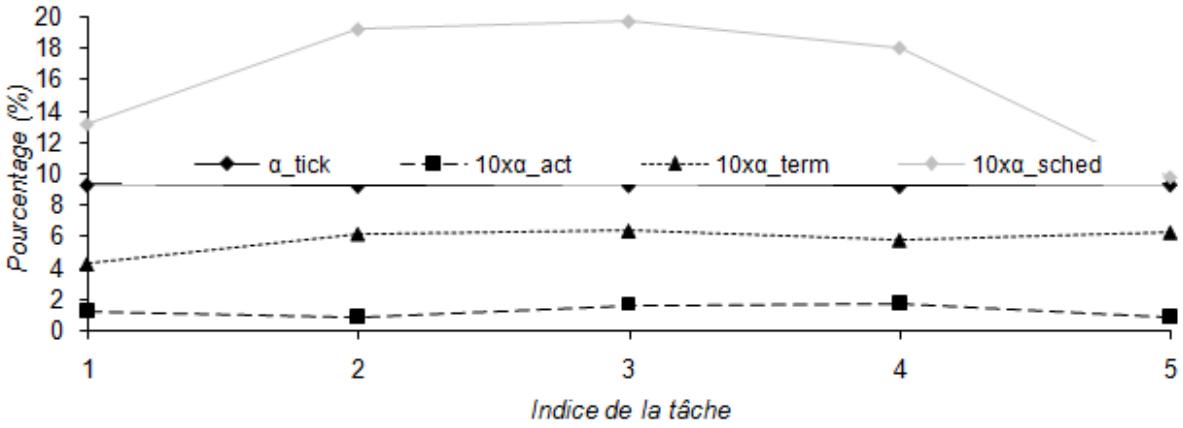

FIGURE 6.24 – Résultats avec EDF : Pourcentages α_{tick}, α_{act}, α_{sched}, α_{term}, α_{get} et α_{rel}

6.c Synthèse

Tout au long de ce chapitre, nous avons constaté, pour les deux politiques d'ordonnancement FP/FIFO et EDF, que les temps de réponse pires cas, calculés sans tenir compte des charges du noyau, sont toujours inférieurs aux temps de réponse réels mesurés sur notre plateforme. Au contraire, l'intégration de ces charges aboutit à des temps de réponse pires cas théoriques toujours supérieurs aux cas réels. Ainsi, les charges dues au système d'exploitation ne peuvent être négligées dans les conditions de faisabilité. De plus, l'écart entre le temps de réponse pire cas théorique considérant les charges du noyau et le temps de réponse réel n'excède jamais 11.41%. Ainsi, ce résultat s'avère plus économique que la méthode empirique, couramment utilisée, qui consiste à réserver 30% de la charge du processeur pour l'exécution du système d'exploitation. En outre, nous avons constaté l'optimalité de l'ordonnancement EDF qui parvient à ordonnancer un ensemble de tâches périodiques alors que l'algorithme d'Audsley échoue.

Chapitre 7

Conclusion et perspectives

Ce chapitre clôture ce rapport de thèse qui traite de la prise en compte de l'exécution du système d'exploitation lors du dimensionnement temps réel d'applications embarquées. Cette étude a été menée sur un système d'exploitation conforme au standard OSEK, véritable référence dans le milieu automobile. Au fil de ce rapport, nous avons présenté les méthodes classiques de dimensionnement temps réel, c'est à dire ne tenant pas compte des charges dues au système d'exploitation, les caractéristiques de tout exécutif OSEK ainsi que les charges de notre noyau. Puis nous avons amélioré ces conditions de faisabilité classiques pour les ordonnancements FP/FIFO et EDF. La section 7.a résume les différents chapitres de ce rapport. A la section 7.b, nous récapitulons les résultats obtenus et leur intérêt pour le dimensionnement temps réel d'applications basées sur un noyau OSEK. Des axes de développement de cette étude sont ensuite évoqués à la section 7.c. Enfin, l'ensemble des publications, auxquelles ce travail a donné lieu, sont listées à la section 7.d.

7.a Résumé des chapitres

Le chapitre 2 rappelle que le dimensionnement temps réel d'une application consiste à calculer, avant son déploiement, le temps de réponse pire cas de chacune des tâches qui la composent. Une fois le système dimensionné, nous pouvons vérifier si celui-ci répond aux contraintes temporelles qui lui sont imposées, c'est à dire si le temps de réponse pire cas de chaque tâche est inférieur ou égal à l'échéance fixée par le concepteur. Ce chapitre présente donc les techniques classiques pour le dimensionnement d'un système temps réel. Les différents modèles, les concepts ainsi que les notations usuels y sont rappelés. Afin de garantir un dimensionnement valable pour tout scénario possible, nous avons opté pour une approche pire cas. Autrement dit, le système sera validé dans les pires conditions pour le respect des contraintes temporelles selon les scénarios dits pires cas. Les principaux algorithmes d'ordonnancement ainsi que les scénarios pires cas correspondant sont également détaillés dans ce chapitre. Enfin, ce chapitre présente les tests de validation à proprement parler, également appelés conditions de faisabilité, adaptés aux ordonnancements FP, FIFO, FP/FIFO et EDF.

Le chapitre 3 présente le standard OSEK qui a vu le jour en 1993 sous l'impulsion de plusieurs industriels automobiles dont BMW, Bosch, DaimlerChrysler, Opel, Siemens, et Volkswagen, mais aussi de l'Université de Karlsruhe. Nous rappelons alors que tout exécutif conforme au standard OSEK est parfaitement statique. Autrement dit, tous les objets du système, comme les tâches et les ressources, sont définis lors de sa conception. Aucune création ou destruction dynamique d'objets n'est possible. De même, le standard OSEK n'autorise aucune priorité dynamique. Ce chapitre expose les différentes politiques d'ordonnancement, la gestion des tâches, la méthode du plafond de priorité et les alarmes mises en oeuvre par ce standard. Enfin, nous y présentons les différentes charges dues au système d'exploitation OSEK ainsi que leurs durées d'exécution pires cas.

Au chapitre 4, nous présentons notre analyse temps réel pour un système monoprocesseur basé sur une implémentation du standard OSEK. Cette analyse peut être utilisée avec tout ensemble de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes, non concrètes et se partageant, au plus, une ressource. Notre objectif consistant à calculer précisément le temps de réponse pire cas de chaque tâche ainsi que le facteur d'utilisation du processeur, nous montrons comment tenir compte des charges dues au système d'exploitation OSEK dans ces calculs. Notre étude est également comparée à d'autres travaux de recherche visant à améliorer les conditions de faisabilité. Enfin, nous terminons ce chapitre en expliquant comment traiter les tâches sporadiques et les interruptions.

Le chapitre 5 concerne la politique d'ordonnancement EDF qui, rappelons le, se base sur des priorités dynamiques. Cette politique s'avère plus optimale que la politique FP/FIFO, prescrite par le standard OSEK et basée sur des priorités fixes. Autrement dit, tout ensemble de tâches ordonnançable avec la politique FP/FIFO l'est obligatoirement avec la politique EDF alors que l'inverse est faux. C'est pourquoi, nous expliquons, dans ce chapitre, comment mettre en oeuvre cette politique en nous appuyant sur les seules tâches à

priorité fixe autorisées par le standard OSEK. Nous détaillons alors les charges cumulées du noyau et de notre algorithme. Après un rappel des notations temps réel nécessaires à l'analyse de la politique EDF, les calculs des temps de réponse pires cas sont détaillés. Cette analyse se limite aux ensembles de tâches à échéance arbitraire, périodiques, préemptives ou non, indépendantes, non concrètes et n'utilisant aucune ressource.

Enfin, le chapitre 6 rend compte des expérimentations menées sur nos conditions de faisabilité, améliorées par la prise en compte du coût du système d'exploitation, pour les politiques d'ordonnancement FP/FIFO puis EDF. Nos conditions de faisabilité pour l'ordonnancement FP/FIFO sont mises en oeuvre sur un ensemble de dix tâches périodiques avec dix valeurs différentes de T_{tick} comprises entre $100\mu s$ et $1000\mu s$. Puis nos conditions de faisabilité pour la politique d'ordonnancement EDF sont expérimentées sur un ensemble de cinq tâches périodiques. L'ensemble de tâches précédent est également testé avec un ordonnancement FP/FIFO afin de montrer que cette politique est bien dominée par l'ordonnancement EDF.

7.b Résultats obtenus

Lors des expérimentations, pour chaque test, réalisé sur un ensemble de tâches donné, des mesures ont été effectuées sur une plateforme réelle. Afin d'obtenir des mesures cohérentes par rapport à nos conditions de faisabilité, nous avons toujours reproduit, autant que possible, les scénarios pires cas. De plus, pour chaque test, nous avons calculé les temps de réponse pires cas théoriques en tenant compte du coût du système d'exploitation puis en l'ignorant. Concernant les ensembles de tâches ordonnancées selon la politique FP/FIFO, les résultats montrent clairement que les charges dues au système d'exploitation ne peuvent être négligées. En effet, les temps de réponse pires cas théoriques, calculés en négligeant l'exécution du noyau, sont notablement inférieurs aux temps de réponse pires cas réels. Au contraire, la prise en compte du système d'exploitation amène des temps de réponse pires cas théoriques toujours supérieurs aux temps de réponse pires cas réels. De plus, l'écart observé ne dépasse jamais 11.41% ce qui permet un choix judicieux de la cible. Concernant la politique EDF, nous avons constaté, là aussi, que les charges dues au système d'exploitation ne peuvent être négligées et que nos temps de réponse pires cas théoriques tenant compte de celles-ci apparaissent toujours, comme dans le cas FP/FIFO, légèrement supérieurs aux temps de réponse réels. De plus, nous avons illustré l'optimalité de cette politique à travers un ensemble de tâches non ordonnançable avec la politique FP/FIFO. Finalement, nos conditions de faisabilité, améliorées en considérant les charges dues au système d'exploitation, s'avèrent utiles et valides pour le dimensionnement temps réel d'applications basées sur un noyau OSEK.

7.c Perspectives

Selon les résultats obtenus, nos conditions de faisabilité intégrant le coût du système d'exploitation se révèlent bénéfiques au dimensionnement temps réel d'applications basées sur un noyau OSEK monoprocesseur. A l'issue de cette thèse, nous envisageons les perspectives suivantes :

- L'intégration d'un contrôle d'admission en-ligne, dans l'exécutif OSEK, nous semble importante pour offrir une plus grande robustesse temporelle aux systèmes réels. Le contrôle d'admission pourrait par benchmark recalculer les temps d'exécution pires cas des tâches pour une adaptation plus précise aux conditions réelles d'exécution.

- L'évolution des processeurs, ces dernières années, s'est accompagnée d'une augmentation de leur fréquence de fonctionnement et, par conséquent, de leur consommation d'énergie. C'est pourquoi, à l'heure actuelle, de nombreux travaux sur la minimisation de cette consommation sont menés [21] [47]. Aussi, nos algorithmes de dimensionnement actuels ne prenant en compte que des contraintes d'échéance de terminaison au plus tard, l'intégration d'autres contraintes comme la consommation d'énergie semble intéressante.

- Les architectures multiprocesseur représentent un autre axe de recherche très actuel. L'intérêt de ces architectures repose sur leur capacité à exécuter plusieurs tâches en parallèle. Une extension des conditions de faisabilité en contexte multiprocesseur [6], tenant compte du coût du système d'exploitation, paraît importante.

- Le consortium Autosar, regroupant les acteurs majeurs du monde de l'automobile, spécifie des mécanismes de robustesse temporelle à implémenter dans tout exécutif Autosar. Actuellement, en cas de déviation par rapport aux spécifications, une alarme est levée mais aucune solution n'est spécifiée. Une solution permettant de mieux anticiper une déviation par rapport aux spécifications, en déterminant précisément les temps de réponse, grâce à la prise en compte du coût du système d'exploitation, comblerait cette lacune.

- Dans toute automobile récente, les processeurs sont interconnectés et forment ainsi un système distribué. De tels systèmes sont également courants dans d'autres domaines que celui de l'automobile. C'est pour cette raison que le standard OSEK comporte une couche réseau [48]. Une extension de nos travaux au contexte distribué permettrait de mieux maîtriser les temps de réponse de bout en bout.

- Enfin, bien qu'effectuée sur un exécutif OSEK, l'analyse temps réel présentée lors de cette thèse doit pouvoir s'appliquer à d'autres exécutifs temps réel.

7.d Liste des publications

– **Revue internationale avec comité de lecture :**

F.Bimbard, and L.George. "Real-Time Analysis to Ensure Deterministic Behavior in a Modular Robot Based on an OSEK System". International Transactions on Systems Science and Applications (ITSSA'06), Volume 2, Number 2, Pages 185-190, 2006.

– **Conférences internationales avec comité de lecture :**

F. Bimbard, L. George, and M. Cotsaftis. "Design of Autonomous Modules for Self-Reconfigurable Robots". 3rd International Conference on Computing, Communications and Control Technologies (CCCT'05), pp 154-158, July 2005, Austin, USA.

F.Bimbard, and L.George. "Feasibility Conditions with Kernel Overheads for Mixed Preemptive and Non-Preemptive Periodic Tasks with FP/FIFO Scheduling on an Event Driven OSEK System". WIP Session, 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06). April 2006, San Jose, California, USA.

F.Bimbard, and L.George. "FP/FIFO Feasibility Conditions with Kernel Overheads for Periodic Tasks on an Event Driven OSEK System". 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06). Pages 566-574, April 2006, Gyeongju, Korea.

F.Bimbard, and L.George. "Feasibility Conditions with Kernel Overheads for Periodic Tasks with Fixed Priority Scheduling on an Event Driven OSEK System". 14th International Conference on Real-Time and Network Systems (RTNS'06). May 2006, Poitiers, France.

F.Bimbard, and L.George. "On the Conception of an Autonomous and Modular Robot Based on an Event Driven OSEK System with Deterministic Real-Time Behavior". IEEE International Conference on Autonomic and Autonomous Systems (ICAS'06). July 2006, Silicon Valley, USA.

F.Bimbard, and L.George. "Real-Time Analysis to Ensure Deterministic Behavior in a Modular Robot based on an OSEK System". IEEE International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS'06). September 2006, Erfurt, Germany.

F.Bimbard, and L.George. "Feasibility Conditions with Kernel Overheads for Mixed Preemptive FP/FIFO Scheduling with Priority Ceiling Protocol on an Event Driven OSEK System". WIP Session, 12th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'07). September 25-28, 2007, Patras, Greece.

– **Ecole d'été :**

F. Bimbard et L. George. "Conditions de Faisabilité Tenant Compte des Charges Occasionnées par un Noyau OSEK pour un Ordonnancement Mixte Préemptif et Non-Préemptif avec Mécanisme de Plafond de Priorité". Ecole d'été Temps Réel (ETR'07). Nantes, Septembre, 2007.

Bibliographie

[1] S. Aldarmi and A. Burns. Time-cognizant value functions for dynamic real-time scheduling. *Technical Report YCS-306, Real-Time Research Group, Department of Computer Science*, The University of York, U.K.

[2] C. Angelov, I. Ivanov, and I. Haratcherev. Schedulability analysis of real-time systems under dual-priority scheduling. *International Conference on Automatics and Informatics*, 2000.

[3] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. *Dept. Comp. Science Report YCS 164, University of York*, 1991.

[4] N. C. Audsley, A. Burns, R. I. David, K. W. Tindell, and A. J. Welling. Fixed priority pre-emptive scheduling : an historical perspective. *The journal of Real-Time Systems*, Vol. 8 pp. 173-198, March/May 1995.

[5] S. Baruah. Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks. *Real-Time Systems*, Vol. 24(1), pp. 93-128, Jan. 2003.

[6] S. Baruah and N. Fisher. The feasibility analysis of multiprocessor real-time systems. *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages pp. 85–96, 2006.

[7] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems*, Vol. 2, pp. 301-324, 1990.

[8] S. Baruah, A. K. Mok, and L. Rosier. Preemptively scheduling hard real-time sporadic tasks on one processor. *Proceedings of the 11th Real-Time Systems Symposium*, pp. 182-190, 1990.

[9] F. Bimbard and L. George. Feasibility conditions with kernel overheads for mixed preemptive fp/fifo scheduling with priority ceiling protocol on an event driven osek system. *12th IEEE international Conference on Emerging Technologies and Factory Automation (ETFA'07)*, September 2007.

[10] F. Bimbard and L. George. Fp/fifo feasibility conditions with kernel overheads for periodic tasks on an event driven osek system. *Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, pages Vol. 0, pp. 566 – 574, April Gyeongju, Korea, 2006.

[11] F. Bimbard and L. George. Feasibility conditions with kernel overheads for periodic tasks with fixed priority scheduling on an event driven osek system. *14th Intern Conference on Real-Time and Network Systems*, May Poitiers, France, 2006.

[12] F. Bimbard and L. George. Feasibility conditions with kernel overheads for mixed preemptive and non-preemptive periodic tasks with fp/fifo scheduling on an event driven osek system. *12th IEEE Real-Time and Embedded Technology and Applications Symposium*, April San Jose, California, United States, 2006.

[13] Bosh. Can specification version 2.0.

[14] A. Burns, D. Prasad, A. Bondavalli, et al. The Meaning and Role of Value in Scheduling Flexible Real-Time Systems. *Journal of Systems Architecture*, Vol. 46, p. 305-325, 2000.

[15] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering*, Vol. 21, No. 5 :475–480, May.

[16] A. Burns and A. Wellings. Engineering a hard real-time system : From theory to practice. *Software Practice and Experience*, 25(7), pp. 705-726, 1995.
[17] G. C. Buttazo. Hard real-time computing systems. *Kluwer Academic Publishers*, 1997.
[18] L. Consortium. Lin specification version 2.1.
[19] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. Ordonnancement temps réel. *Edition Hermes Science*, Janvier 2000.
[20] M. Dertouzos. Control Robotics : the procedural control of physical processors. *Proceedings of the IFIP congress*, pp. 807-813, 1974.
[21] B. Gaujal and N. Navet. Dynamic voltage scaling under edf revisited. *Real-Time Systems*, Vol. 37, No. 1 :pp. 77–97, 2007.
[22] L. George. Conditions de faisabilité pour l'ordonnancement temps réel préemptif et non préemptif. *Ecole d'été sur le Temps Réel, ETR'2005, ENSG, Nancy*, Septembre.
[23] L. George. Ordonnancement en-ligne temps réel critique dans les systèmes distribués. *Thèse de doctorat en informatique*, Université de Versailles St-Quentin, 26 janvier 1998.
[24] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive scheduling real-time uniprocessor scheduling. *INRIA Research Report*, No. 2966, September 1996.
[25] C. Han and H. Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. *Real-Time Systems Symposium*, pp. 36-45, 1997.
[26] J. Jackson. Scheduling a production line to minimize maximum tardiness. *Tech. rep. University of California*, Report 43, 1955.
[27] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. *IEEE Real-Time Systems Symposium*, pp. 129-139, San Antonio, USA, December 1991.
[28] K. Jeffay and D. Stone. Accounting for interrupt handling costs in dynamic priority task systems. *Real-Time Systems Symposium*, pp. 212-221, Dec. 1993.
[29] M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Comp. Jour.*, 29(5), pp. 390-395,, 1986.
[30] D. Katcher, J. Lehoczky, and J. Strosnider. Scheduling models of dynamic priority schedulers. *Research Report CMUCDS-93-4*, Carnegie Mellon University, Pittsburgh, April 1993.
[31] G. Le Lann. A Methodology for Designing and Dimensioning Critical Complex Computing Systems. *IEEE Symposium and Workshop on Engineering of Computer Based Systems*, pp. 332-339, March 1996.
[32] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proceedings 11th IEEE Real-Time Systems Symposium*, pp 201-209, Dec. Lake Buena Vista, FL, USA, 1990.
[33] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm : exact characterization and average case behavior. *Real-Time Systems Symposium*, pp. 166-171, Dec. 1989.
[34] W. Lei, W. Zhaohui, and Z. Mingde. Worst-case response time analysis for osek/vdx compliant real-time distributed control systems. *28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, pages pp. 148–153, 2004.
[35] J. Y. T. Leung and M. Merril. A note on premptive scheduling of periodic, Real Time Tasks. *Information Processing Letters*, Vol. 11(3) pp. 115-118, Nov. 1980.
[36] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4) pp. 237-250, December 1982.
[37] J. Liu. *Real-time systems.* Prentice Hall, 2000.
[38] L. C. Liu and W. Layland. Scheduling algorithms for multiprogramming in a hard real time environment. *Journal of ACM*, Vol. 20, No 1, pp. 46-61, January 1973.
[39] S. Martin. Maîtrise de la dimension temporelle de la qualité de service dans les réseaux. *Ph.D. Thesis*, University of Paris 12, July 2004.
[40] S. Martin, P. Minet, and L. George. Improving non-preemptive Fixed Priority scheduling with Dynamic Priority as secondary criterion. *RTS'05*, Paris, France, April 2005.

[41] P. Meumeu Yomsi and Y. Sorel. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. *Proceedings of the 19th EuroMicro Conference on Real-Time Systems (ECRTS'07), IEEE Computer Society Press*, July.
[42] P. Meumeu Yomsi and Y. Sorel. Schedulability analysis using exact number of preemptions and no idle time for real-time systems with precedence and strict periodicity constraints. *Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS'07)*, March.
[43] A. K. Mok. Fundamental design problems for the hard real-time environments. *MIT Ph.D. Dissertation*, May 1983.
[44] A. K. Mok and M. L. Dertouzos. Multiprocessor scheduling in a hard real-time environment. *Proc. Seventh Texas Conf. Comput. Syst.*, November.
[45] P. Muhlethaler and K. Chen. A Scheduling Algorithm for Taks Described by Time Value Functions. *Real Time Systems*, Volume 10, number 3, May 1996.
[46] N. Navet. *Evaluation de performances temporelles et optimisation de l'ordonnancement de tâches et messages.* Thèse de doctorat de l'Institut National Polytechnique de Lorraine, Novembre 1999.
[47] N. Navet and B. Gaujal. Ordonnancement temps réel et minimisation de la consommation d'énergie. *Chapter 4, Traité I2C Systèmes Temps Réel volume 2, Hermès Science, ISBN2-7462-1304-4*, June 2006.
[48] OSEK/VDX. Osek com specification version 3.0.3.
[49] OSEK/VDX. Osek operating system specification version 2.2.
[50] P. Puschner and A. Schedl. Computing maximum task execution times – a graph-based approach. *Journal of Real-Time Systems*, 13(1) :67–91, Jul. 1997.
[51] S. Seo, S. Lee, S. Hwang, and J. Jeon. Analysis of task switching time of ecu embedded system ported to osek(rtos). *SICE-ICASE International Joint Conference*, pages 18–21, October Bexco, Busan, Korea, 2006.
[52] M. Spuri. Analysis of deadline scheduled real-time systems. *INRIA Research Report*, No. 2772, Jan. 1996.
[53] A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. Deadline scheduling for real-time systems. *Kluwer Academic Press*, 1998.
[54] D. B. Stewart and P. K. Khosla. Real-time scheduling of sensor-based control systems. 8^{th} *IEEE workshop on real-time operating systems*, Vol. 20 pp. 139-144, May 15-17 Atlanta, USA, 1991.
[55] K. Tindell, A. Burns, and A. J. Wellings. Analysis of hard real-time communications. *Real-Time Systems*, Vol. 9, pp. 147-171, 1995.
[56] H. Tokuda and C. Mercer. ARTS : A Distributed Real-Time Kernel. *Operating systems review*, Vol. 23 (3), p. 29-53, July 1989.