REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE AKLI MOAND OULHADJE-BOUIRA

Faculté des Sciences et des Sciences Appliquées
Département d'Informatique

Mémoire de fin d'étude

Présenté par :

ZENATI Fatima

RECHAM Ouardia

En vue l'obtention du diplôme de **Master 02** en :

Filière : **INFORMATIQUE**

Option : **Ingénierie Des Systèmes D'information Et Logiciels**

Thème :

Ordonnancement Temps Réel Régulé Sous Contrainte Des Ressources Critiques Et D'énergie Renouvelable

Devant le jury composé de :

BOUDJELABA Hakim	MAA	UAMOB	Président
ABBAS Akli	MCB	UAMOB	Encadreur
HAMID Rabah	MAA	UAMOB	Examinateur

Année Universitaire 2017/2018

Remerciements

Nous tenons avant tout à remercier Dieu tout puissant de nous avoir donné la force et la volonté pour achever ce modeste travail. Nous tenons à remercier particulièrement notre encadreur Mr ABBAS Akli, pour nous avoir dirigés dans notre travail et pour son prestigieux aide, sa disponibilité et avis éclairés.

Un grand merci à tout qui nous a aidés dans le département informatique et également à tous les enseignants pour tout le savoir qu'ils ont su nous transmettre durant ces dernières années. Et enfin nous remercions tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Dédicaces

Je tiens à dédier ce modeste travail :

A mes chers Parents, ma Sœur et mes Frères ;

A mon Marie pour leur plus grand encouragement de la patience et de l'aide ;

A la lumière de ma vie mes Enfants ;

A toute la Famille et mes Amis.

ZENATI Fatima.

Dédicaces

Je tiens à dédier ce modeste travail :

A mes chers parents ;

A mes Sœurs ;

A mes Frères.

RECHAM Ouardia.

Résumé

Dans notre travail nous avons traité le problème d'ordonnancement temps réel régulé sous contraintes d'énergie renouvelable et de ressources critiques.

Un système temps réel est souvent un système multitâche incluant un gestionnaire de tâches (ordonnanceur). L'ordonnancement est un point crucial des systèmes temps réel ; en effet l'ordonnancement va déterminer le comportement temporel et être le garant du respect des contraintes (de temps, d'énergie et de ressources critiques) imposées à l'exécution de l'application. L'ordonnancement temps réel régulé consiste à appliquer le principe de régulation à boucle fermée (commande par rétroaction) sur des systèmes informatiques dans le but d'améliorer leurs performances pendant leur exécution.

La solution que nous avons proposée dans notre contribution permet d'exécuter un ensemble de tâches périodiques partageant des ressources critiques, sur un processeur à vitesse variable alimenté par une batterie qui est rechargée par une énergie renouvelable (énergie solaire). L'objectif est d'obtenir un ordre d'exécution des tâches qui garantit le respect de partage de ressources à moins consommation d'énergie.

Mots clés :Système embarqué - Ordonnancement temps réel - Ordonnancement temps réel régulé - Partage de ressources - Tâches périodiques - Réduction de consommation - Récupération de l'énergie.

Abstract

In our work we have dealt with the problem of real-time scheduling regulated under constraints of renewable energy and critical resources.

A real-time system is often a multitasking system that includes a task manager (scheduler). Scheduling is a crucial point of real-time systems ; Indeed, the scheduling will determine the temporal behavior and be the guarantor of the respect of the constraints (time, energy and critical resources) imposed on the execution of the application. Regulated real-time scheduling involves applying closed-loop control (feedback control) to computer systems to improve performance during execution.

The solution we proposed in our contribution allows to perform a set of periodic tasks sharing critical resources, on a variable speed processor powered by a battery that is recharged by renewable energy (solar energy). The goal is to obtain a task execution order that guarantees the respect of sharing resources with less energy consumption.

Key words : Embedded system - Real-time scheduling - Real-time Feedback scheduling - Resource sharing - Periodic tasks - Consumption reduction - Energy recovery.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

L'évolution de la technologie favorise l'insertion de l'informatique dans la majorité des objets qui nous entourent, en particulier les systèmes temps réel qui sont souvent utilisés dans un contexte embarqué. Aujourd'hui, les systèmes embarqués prennent une place importante dans notre vie quotidienne. Ils sont variés et apparaissent dans des secteurs extrêmement divers tel que le transport, la médecine, le multimédia, les téléphones mobiles...

Les systèmes embarqués sont autonome et interdisent les interventions humaines parce qu'ils sont inaccessibles ou déployés en trop grand nombre. Ils sont alimentés par des batteries ou supercondensateurs avec une quantité d'énergie limitée et peuvent être épuisés, ce qui implique la fin de fonction de tels systèmes. Pour étendre la durée de fonctionnement de ce système, l'utilisation des énergies renouvelables (exemple de l'énergie solaire) se révèle être une alternative de choix pour alimenter un bon nombre de système.

Le problème de la gestion de l'énergie dans les systèmes embarqués revient à minimiser sa consommation pour assurer l'autonomie et le fonctionnement durable de ces systèmes. Ce problème est généralement traité par la méthode DVFS (Dynamic Voltage Frequancy Scaling). Ainsi, les systèmes temps réel embarqués sont caractérisés par une forte interaction avec les procédés contrôlés. Le comportement concurrent des événements externes amène à décrire l'environnement comme un système fortement parallèle. Pour cela, l'architecture adéquate pour répondre à ce comportement parallèle est une architecture multitâche. Ces différentes tâches peuvent être liées par des relations de synchronisation, de communication et de partage de ressources. En plus, ces systèmes sont généralement critiques en terme de temps, ce qui implique des contraintes temporelles à respecter.

Le problème d'ordonnancement temps réel des différentes tâches sous diverses contraintes dans les systèmes embarqués a été largement traité. Toute fois, les travaux réalisés ne prennent pas en considération l'ordonnancement des tâches périodique sous contraintes d'énergie et de ressources critiques. L'ordonnancement classique soufre de plusieurs limites. A cet effet, de nombreux chercheurs ont proposé d'utiliser la théorie de commande par rétroaction pour remédier à ces limites. L'objectif est d'ajuster les paramètres d'ordonnancement du système en fonction de certaines mesures.

Dans notre contribution nous avons traité le problème d'ordonnancement temps réel régulé sous contraintes d'énergie renouvelable et de ressources critiques. La solution que nous avons proposée vise à prendre en considération les incertitudes des durées d'exécution des tâches périodique, la quantité d'énergie disponible et les ressources partagés. L'objectif est d'améliorer l'ordonnancement et d'optimiser la consommation énergétique.

Notre mémoire est structuré comme suit :

Chapitre 1 : Nous introduirons les systèmes temps réel, la modélisation des tâches temps réel, l'ordonnancement temps réel et les solutions proposées dans la littérature.

Chapitre 2 : Nous présentons les limites d'ordonnancement temps réel classique, les stratégies de commande des procédés, la problématique d'ordonnancement basé sur WCET (Worst Case Execution Time), l'ordonnancement temps réel régulé et un état de l'art des travaux portant sur l'ordonnancement régulé.

Chapitre 3 : Nous présentons l'énergie renouvelable, la consommation énergétique dans les systèmes temps réel, les méthodes de réduction de la consommation énergétique et les solutions proposées pour l'ordonnancement temps réel sous contraintes d'énergie renouvelable.

Chapitre 4 : Nous détaillerons notre contribution dans l'ordonnancement temps réel régulé des tâches périodiques sous contraintes d'énergie renouvelable et de ressources critiques et l'évaluation de la solution proposée.

Première partie

L'état de l'art

Chapitre 1

Généralités sur les systèmes temps réel

1.1 Introduction

Pendant ces dernières années, les systèmes temps réel sont très largement utilisés, on les trouve dans des applications extrêmement variées comme le système de freinage ABS, les produits technologique largement consommation (smartphone, jeux), les systèmes de transport (trains, automobiles), aériens (avions), spatiaux (satellites, navettes, fusées). Un système temps réel par rapport à un système informatique conventionnel, n'est pas nécessairement un système qui va vite mais celui qui satisfait à des contraintes temporelles imposées par l'application dans laquelle il est mis en œuvre. En effet, cette dynamique peut être de l'ordre de la milliseconde pour les systèmes radar, de la fraction de seconde pour les systèmes multimédias, de la minute pour le contrôle de production, de l'heure pour la prévision météo.

1.2 Définition

1.2.1 Système temps réel

Un système temps réel est un système (application ou un ensemble d'applications) informatique qui doit répondre à des stimuli fournis par un environnement externe afin de le contrôler. En outre, dans tous les cas même les pires, toutes les contraintes temporelles doivent être respectées sans quoi le système est défaillant. La correction d'un tel système dépend non seulement de la justesse des calculs mais aussi du temps auquel la réponse est fournie (contraintes temporelles).

En effet, plusieurs définitions ont été proposées pour préciser la notion du système temps réel « Un système est dit temps réel si l'exactitude des applications ne dépend pas seulement du résultat mais aussi du temps auquel ce résultat est produit » [2].
Une autre définition « un système est dit temps réel s'il doit s'exécuter suffisamment vite par rapport à la dynamique du procédé contrôlé. La notion relative de vitesse s'exprime par des contraintes temporelles » [2].

1.2.2 Système contrôle commande

Est un système informatique de contrôle de procédé où ce dernier désigne un système physique contrôlé[2].

Afin de contrôler le procédé, le système informatique est en relation avec l'environnement physique externe par l'intermédiaire des capteurs (sensors) qui permettent d'obtenir des informations sous la forme des interruptions (information tout ou rien) ou des mesures (information continue), et utilise un calculateur qui traite ces données et, en fonction du résultat, évalue une décision qui agit sur cet environnement extérieur, par l'intermédiaire d'actionneurs (actuators), sous la forme de commandes (modification d'état physique du système) ou sous la forme d'un affichage (diodes, lampes, afficheurs, écrans, etc.)(voir Figure 1.1).

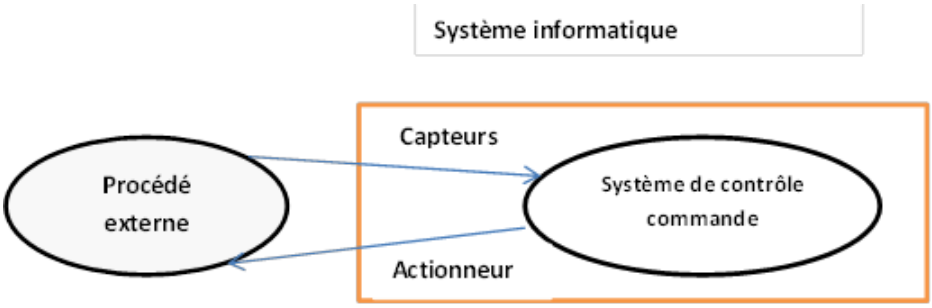

FIGURE 1.1 – Un système de contrôle commande

1.2.3 Système embarqué

Est un système informatique autonome utilisé pour des tâches précises, comportant souvent des contraintes de temps réel, et interagit fortement avec son environnement. Dans

un système embarqué, les moyens de calcul sont embarqués sur le procédé contrôlé. Le fait d'embarquer les moyens de calcul implique, en plus des contraintes d'encombrement (taille, poids, forme), des contraintes de consommation d'énergie puisque l'alimentation électrique des éléments de calcul est soit embarquée (batteries, carburant, etc.), soit ambiante (panneaux solaires, etc.) [2].

Systèmes embarques sont présents dans :

- Automobile
- Avionique et aérospatiale
- Électroménagers
- Électronique domestique (téléviseurs, réseautique, consoles de jeu)
- Téléphones intelligents

Les systèmes embarqués sont utilisés dans des applications de plus en plus critiques dont le dysfonctionnement peut générer des catastrophiques (applications médicales, de transport).

1.3 Architecture des systèmes temps réel

Un système temps réel est constitué d'une couche logicielle s'exécutant sur un ou des calculateur (s) constituant la couche matérielle.

1.3.1 Architecture matérielle

Désigne l'ensemble des ressources matérielles (ou physiques) qui peuvent être nécessaire à l'exécution de la couche logicielle : cela inclut les processeurs, la mémoire, les réseaux, les cartes d'entrées / sorties (qui sont reliées aux capteurs et actionneurs), etc.
Les architectures peuvent être classées en trois catégories selon leur composition, et notamment selon le nombre de processeurs utilisés et selon la présence ou non de réseaux :

1.3.1.1 Architecture monoprocesseur

La couche matérielle est composée d'un seul processeur qui exécute toutes les applications composant le système temps réel, alors les différentes applications partagent le temps processeur. Dans ce type il n'a pas de réseaux.

1.3.1.2 Architecture multiprocesseur

La couche matérielle est composée de plusieurs processeurs partageant une même mémoire centrale. Les applications du système sont donc réparties sur ces différents processeurs. Dans ce type on note l'absence de réseaux.

1.3.1.3 Architecture distribuée

La couche matérielle est constituée de plusieurs processeurs, chaque processeur possède une mémoire propre à lui. Il n'y a donc pas de mémoire commune. Les applications qui se trouvent sur des processeurs différents et ayant besoin de communiquer entre elles le feront à travers un réseau.

Dans notre travail, on considère que l'architecture est monoprocesseur.

1.3.2 Architecteur logicielle :

L'architecture logicielle d'un système temps réel se décompose en deux parties différentes (voir Figure 1.2) :

Un programme applicatif C'est une partie de haut niveau, correspondant aux fonctions permettant de contrôler le système temps réel.

Exécutif temps réel Est de plus bas niveau, joue le rôle d'un système d'exploitation lorsqu'il est soumis à des contraintes de temps, ou bien lorsqu'il doit être embarqué sur un microcontrôleur, et faisant le lien entre la couche matérielle et programmes applicatifs.

1.3.2.1 Programme applicatif

Est la partie logicielle du système qui va exécuter les différentes fonctions nécessaires au contrôle du système, et plus particulièrement du procédé. Le comportement concurrent des événements et grandeurs physiques externes amène à décrire l'environnement (procédé) comme un système fortement parallèle. Pour gérer toutes ces entités qui interagissent, il est donc nécessaire de développer des techniques logicielles permettant de traiter les informations, et de produire les actions appropriées. Aussi, l'architecture la mieux adéquate pour répondre à ce comportement parallèle du procédé est une architecture multitâche. La

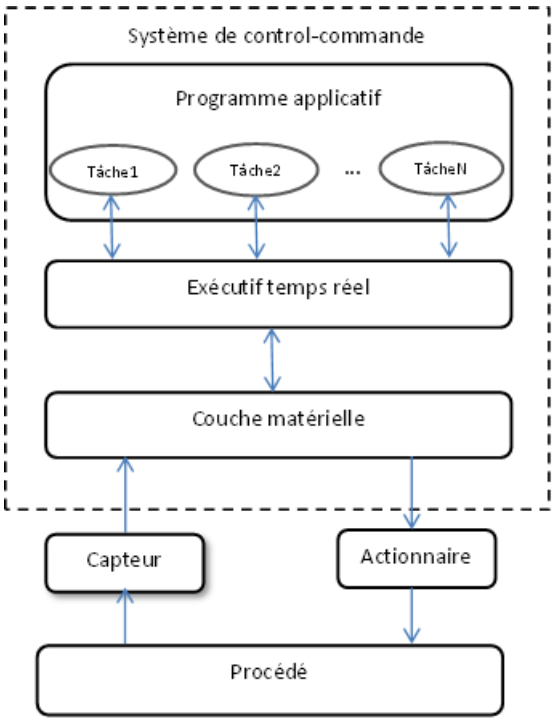

FIGURE 1.2 – Architecture d'un système temps réel

réponse au parallélisme de l'environnement est le parallélisme de conception. Elle permet de faciliter la conception, de réduire le coût de développement par rapport à une application fonctionnant sans parallélisme et surtout augmente l'évolutivité de l'application réalisée. Ainsi, le programme applicatif est divisé en entités distinctes appelées tâches ou processus, chacune ayant un rôle propre comme par exemple[3] :

Les tâches d'entrées/sorties : Ces tâches accèdent aux données par l'intermédiaire de cartes d'entrées/sorties liées au procédé géré.

Les tâches de traitement : Ces tâches intègrent le traitement des événements comme les signaux ou des lois de commande.

Les tâches de gestion de l'interface utilisateur : Ces tâches permettent de présenter l'état du procédé ou de sa gestion à l'utilisateur. En réponse, l'opérateur peut modifier les consignes données ou changer les commandes.

Tâches de communications : Ces tâches s'occupent des messages envoyés ou reçus via un ou plusieurs réseaux. Si ce type de tâches existe, l'application est dite distribuée ou répartie.

Les tâches de sauvegarde : Ces tâches permettent de stocker l'état du système. Cette sauvegarde peut être utilisée lors de la reprise d'exécution ou pour analyser le fonctionnement de l'application.

Interactions entre les tâches : Les tâches obtenues du précédent découpage, et qui constituent le programme applicatif, ne sont pas des entités d'exécution indépendantes. En effet, certaines tâches sont connectées vers l'extérieur pour les entrées/sorties. De plus elles peuvent être liées par des relations de type :

Synchronisation : cela se traduit par une relation de précédence d'exécution entre les tâches ; Communications : à la notion de précédence, traduite par la synchronisation, s'ajoute le transfert de données entre les tâches ;

Partage de ressources : comme des zones mémoire, des cartes d'entrées/sorties, cartes réseau, etc. Certaines de ces ressources ne doivent pas être accessibles par plus d'une tâche à la fois, elles sont dites ressources critiques, et pour avoir un bon fonctionnement de l'application, il est nécessaire de mettre l'accès à ces ressources en exclusion mutuelle.

1.3.2.2 Exécutif temps réel

Le programme applicatif est géré par un système d'exploitation qualifié d'exécutif temps réel. Sa principale caractéristique est son déterminisme d'exécution avec des paramètres temporels fixés (temps de prise en compte d'une interruption, changement de contexte entre deux tâches, etc). L'exécutif temps réel est composé de deux parties :

Le noyau temps réel (kernel, en anglais) : Assure la gestion de tâches (ordonnancement, outils de synchronisation et de communication, gestion des interruptions) et de la mémoire.

Des modules ou bibliothèques venant compléter le noyau temps réel et facilitant l'exécution d'une application à travers l'apport de routines de gestion de fichiers, de gestion de timers, de gestion de réseaux, etc... [1]

Exécutif temps réel possède les services classiques d'un système d'exploitation, mais plus particulièrement :

— Gestion du temps (attente d'un délai)
— Gestion de tâches (création, arrêt)
— Gestion de la mémoire ;
— Une allocation des ressources sans interblocage et inversion de priorité ;
— Un ordonnancement spécifique et certifiable.

Dans cet environnement le point essentiel est la politique d'ordonnancement qui est le mécanisme permettant de choisir la tâche qui va être exécutée par le processeur à un instant donné, afin de respecter l'ensemble des contraintes liées à la gestion du procédé. L'algorithme qui va effectuer ce choix est appelé l'ordonnanceur (scheduler, en anglais). Il existe deux manières d'appeler cet ordonnanceur :

— Appels à intervalle régulier (par exemple à chaque unité de temps) ;
— Appels basés sur des évènements comme l'arrivée, la fin d'exécution ou l'échéance d'un travail.

Les langages de programmation utilisés pour développer une application temps réel s'appuient sur les services fournis par l'exécutif temps réel afin de gérer des tâches, de les faire communiquer, se synchroniser, gérer le temps, et de traiter les interruptions matérielles. Ces services influencent l'état des tâches, sur lequel s'appuie l'ordonnanceur.

1.4 Les principales caractéristiques des systèmes temps réel

Grande diversité des dispositifs d'entrées / sorties : les données à acquérir qui sont fournies par les capteurs et les données à fournir aux actionneurs sont de types très variés (continu, discret, tout ou rien ou analogique)[2].

Prise en compte des comportements concurrents : l'ensemble de données physiques qui arrivent de l'extérieur et le réseau qui permet de recevoir des messages ne sont pas synchronisés au niveau de leurs évolutions, par conséquent, le système informatique doit être capable d'accepter ces variations simultanées des paramètres[2].

Respect des contraintes temporelles : la caractéristique précédente impose de la part du système informatique d'avoir une réactivité suffisante pour prendre en compte tous ces comportements concurrents et en réponse à ceux-ci, de générer une commande en respectant un délai compatible avec la dynamique du système[2].

Sûreté de fonctionnement : les systèmes temps réel de type contrôle-commande mettent souvent en jeu des applications qui demandent un niveau important de sécurité pour raisons de cout de vie humaines. Pour répondre à cette demande, il est nécessaire de mettre en œuvre toutes les réponses de la sûreté de fonctionnement (développement sûres, tests, méthodes formelles, prévisibilité, déterminisme, continuité de service, tolérance aux fautes, redondance, etc.)[2].

1.5 Classification des systèmes temps réel

Dans le contexte temps réel, les résultats valides se caractérisent à la fois par leur exactitude logique et leur exactitude temporelle. Lorsqu'une contrainte temporelle n'est pas respectée (échéance dépassée), les conséquences pourront être, soit catastrophiques, soit juste dommageables à l'application. Selon la gravité qu'entraîne un non-respect de contrainte temporelle (exprimée en termes d'échéance), on distingue trois catégories de système temps réel[4].

1.5.1 Un système temps réel dur (hard-real-time) :

est un système dans lequel la gestion du temps doit être stricte afin de conserver l'intégrité du service rendu et le non-respect des contraintes temporelles peut provoquer des conséquences catastrophiques sur l'environnement contrôlé. L'exemple d'applications concernées est celle de la supervision médicale (voir Figure 1.4).

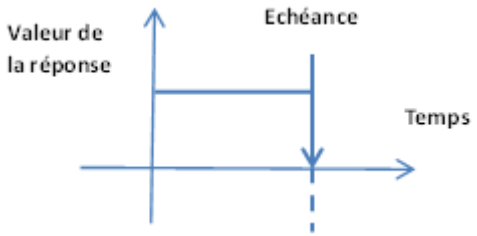

FIGURE 1.3 – Système temps réel dur

1.5.2 Un système temps réel souple (soft-real-time) :

est un système dans lequel le non-respect des échéances ne peut occasionner de graves conséquences, mais peut engendrer une dégradation du service rendu. Exemple du système de projection vidéo (voir Figure 1.4).

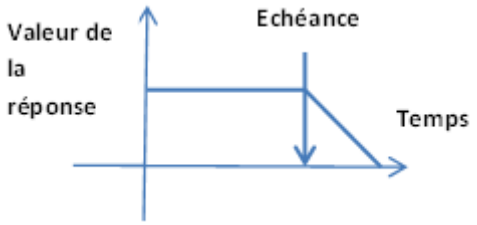

FIGURE 1.4 – Système temps réel souple

1.5.3 Un système temps réel ferme (firm-real-time) :

est un système temps réel souple mais où il n'y a aucun intérêt à avoir du retard ou est un système temps réel dur pour lequel quelques échéances peuvent être occasionnellement manquées (voir Figure 1.5).

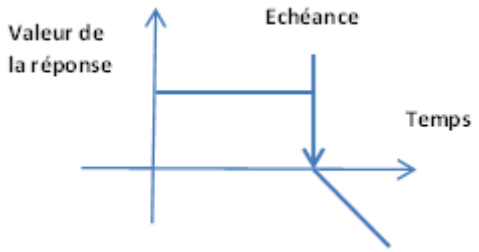

FIGURE 1.5 – Système temps réel ferme

1.6 Tâches temps réel

1.6.1 Définition :

On appelle une tâche (tasks en anglais), une entité d'exécution ou instance dynamique d'un programme exécutable. On peut la définir aussi comme l'exécution d'une suite d'instructions. Cette suite d'instructions peut être ré-exécutée plusieurs fois de façon périodique ou apériodique. Une tâche est une activité qui consomme des ressources de la machine informatique (de la mémoire, du temps CPU et de l'énergie). Une application temps réel est constituée d'un ensemble de tâches.
on appelle travail ou instance (job en anglais) d'une tâche une exécution ou occurrence de celle-ci. Ainsi, une tâche est constituée d'un ensemble infini de travaux.
Dans un environnement multitâche, chaque tâche peut être dans l'un des états suivants :(voir Figure 1.6) :

Prête (ready) : si la tâche est en attente de pouvoir s'exécuter après son réveil.

Active ou Elue(running) : si la tâche est en train de s'exécuter. Dans le cas d'un seul processeur, une seule tâche est en cours d'exécution à un instant donné (selon la politique d'ordonnancement).

Bloquée (en attente) ou (waiting) : c'est le cas des tâches qui sont en attente d'événements qui provoqueront leur réveil (interruption, réception de message . . .).

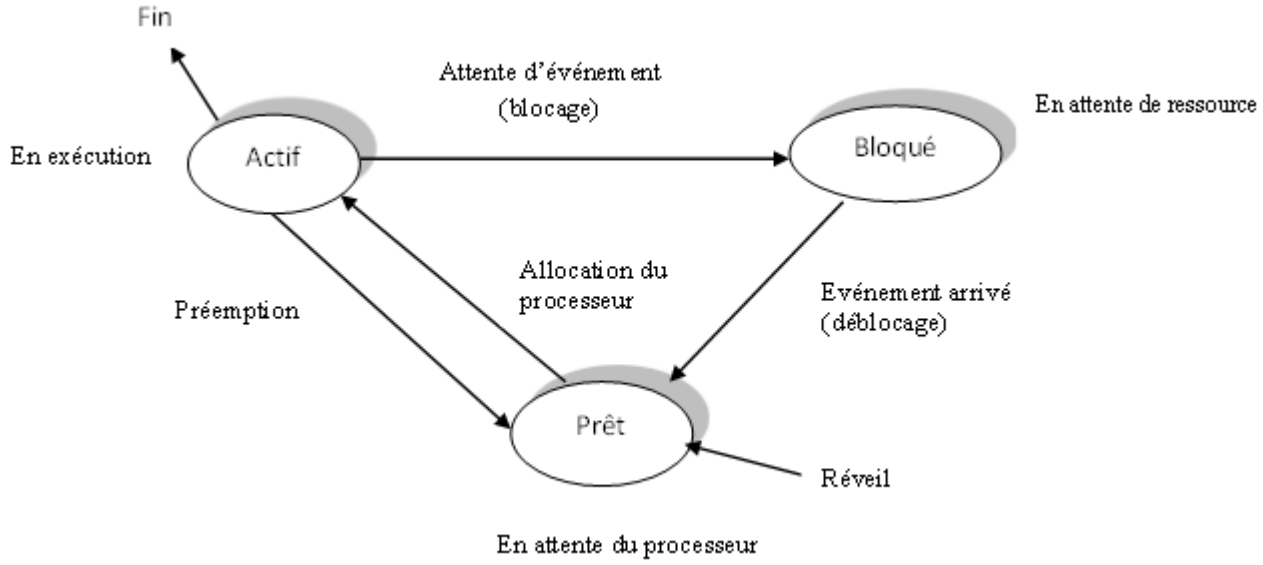

FIGURE 1.6 – États possibles d'une tâche

1.6.2 Caractérisation d'une tâche temps réel

Une application temps réel se définit par l'exécution de traitements spécifiques qui requièrent des besoins en ressources logiques ou matérielles. En outre elle peut être soumise à des contraintes, en plus des contraintes temporelles, liées aux spécifications du système. Parmi celles-ci, on peut citer [5] :

1.6.2.1 Les contraintes de ressources :

le rôle de système d'exploitation est d'arbitrer entre les demandes des tâches de l'application et les ressources effectivement disponibles dans le système. Les tâches partagent des ressources critiques en exclusion mutuelle.

1.6.2.2 Les contraintes de synchronisation :

Des tâches peuvent avoir entre elles des relations de précédence. S'il existe une relation de précédence qui impose un ordre dans lequel les tâches doivent s'exécuter, on dit que les tâches sont dépendantes. Si n'ayant pas sera qualifiée de tâche indépendante

1.6.2.3 Les contraintes d'exécution :

En fonction de contraintes de précédence et de ressources, on peut distinguer deux modes d'exécution de tâches temps réel.

Préemptif : L'exécution d'une tâche peut être interrompue à tout instant lorsqu'une tâche jugée plus urgente est activée. Son exécution est reprise ultérieurement. Si une tâche demande une ressource critique, occupée par une tâche moins prioritaire, elle doit attendre sa libération pour continuer son exécution.

Non préemptif : L'ordonnanceur n'interrompt jamais l'exécution d'une tâche en cours au profit d'une autre tâche. Il doit attendre jusqu'à la fin de l'exécution de la tâche en cours, avant de commencer l'exécution de toute autre tâche. Si une tâche non préemptive est interrompue, son exécution doit être reprise de nouveau.

1.7 Modèles de tâches temps réel :

On distingue plusieurs classes de tâches selon la manière dont les instances sont activées.

1.7.1 Modèle de tâches périodiques :

Sont des tâches récurrentes, s'activent à des intervalles réguliers de temps. L'intervalle de temps entre deux réveils (activations) consécutifs est donc constant ; elles correspondent aux mesures sur le procédé. Il s'agit généralement des taches de suivi de contrôle du procédé. On rencontra ce type de tâches dans la quasi-totalité des applications de contrôle/commande et dans une majorité des applications temps réel.

Les paramètres spécifiques pour le modèle de tâches périodiques sont présentés dans la Figure 1.7

La signification des symboles est la suivante :

r_0 **:** le moment de la première requête d'exécution de la tâche (temps initial)

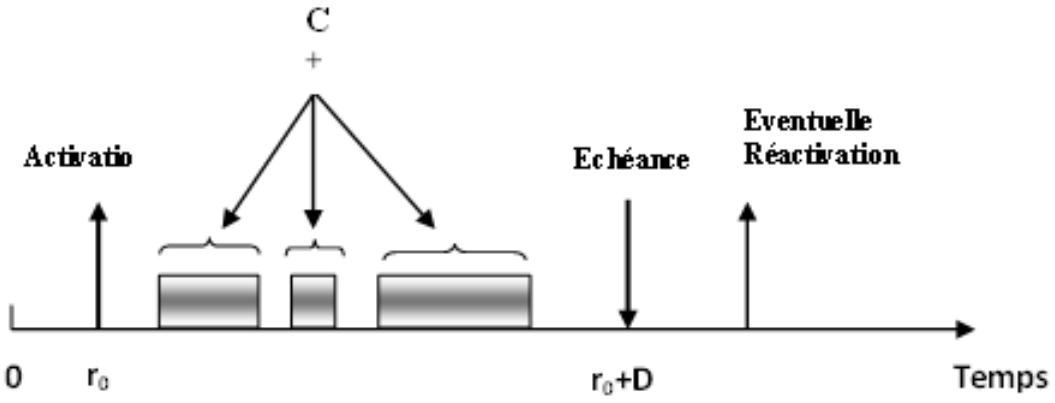

FIGURE 1.7 – Modèle de tâches périodiques

C_i **:** la durée d'exécution maximale de la tâche, quand elle dispose du processeur pour elle (le coût d'exécution), Cette quantité est appelé généralement WCET (Worst Case Execution Time)

D_i le délai critique acceptable pour l'exécution de la tâche (l'échéance),

P_i **:** la période d'exécution, écart entre deux activations successives.

La relation entre période et délai critique a une grande importance lors de l'étude temporelle des systèmes, on distingue trois cas pour une tâche [6] :

Périodiques à échéance sur requête ($D_i = P_i$) l'échéance d'une instance égale à la date d'activation du suivant (voir Figure 1.8).

Périodiques à échéance contrainte ou l'échéance est inférieure ou égale à la période ($D_i \leq Pi\ \forall i$, i $n\circ$ de la tâche), les contraintes temporelles à respecter (voir Figure 1.9).

Périodiques à échéances arbitraires l'échéance supérieure à la période $D_i > P_i$, les contraintes temporelles non respectées de temps à autre.

Un système de tâches périodiques est caractérisé par :

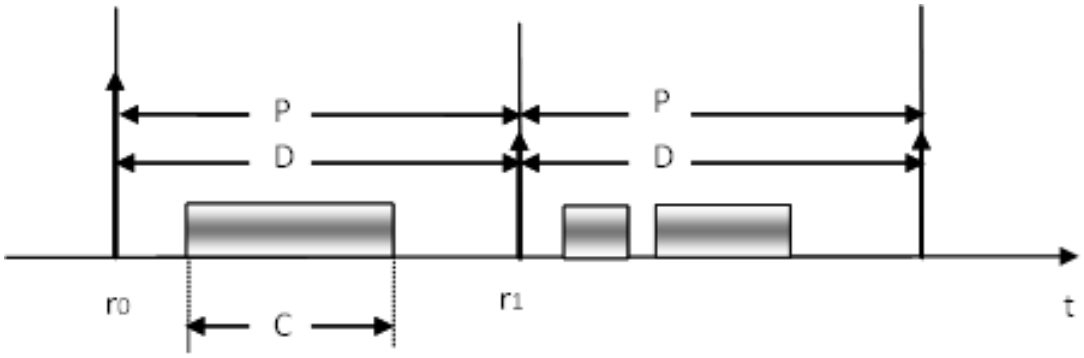

FIGURE 1.8 – Modèle de tâches périodiques à échéance sur requête

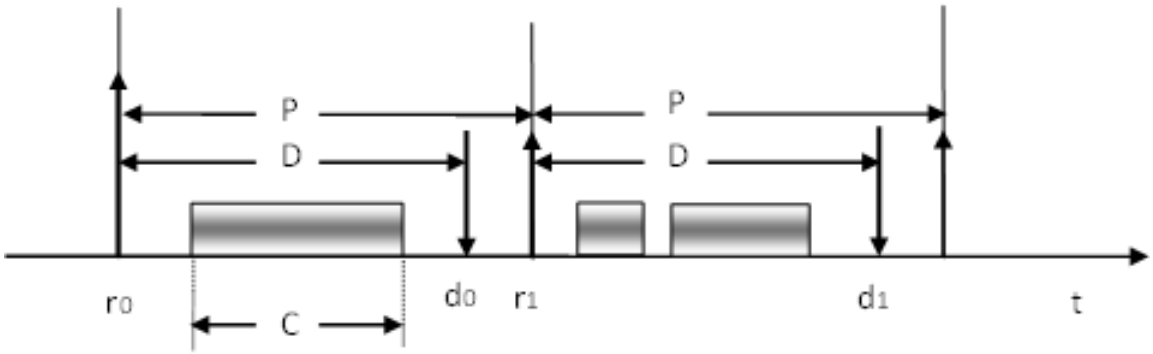

FIGURE 1.9 – Modèle de tâches périodiques à échéance contrainte

— Un facteur d'utilisation U :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \tag{1.1}$$

La date de réveil r_k du la$k^{ème}$ instance est définie par :

$$r_k = r_0 + K * P \tag{1.2}$$

— Et l'échéance dk de la $k^{ème}$ instance est définie par :

$$d_k = r_k + D \tag{1.3}$$

1.7.2 Modèle de tâches apériodiques :

En ce qui concerne les tâches apériodiques elles sont réveillées à des instants imprévisibles par l'arrivée des évènements qui peuvent être produit à tout instant, ces évènement sont

extérieur au système ou lie à un signal logiciel émis par une autre tâche au cours de son exécution. La tâche apériodique est caractérisés par un seul paramètre connu est la durée d'exécution C.

On peut représenter l'exécution d'une tâches apériodique dans le diagramme de Gantt comme montre dans la Figure 1.10. (r et r') sont deux dates de réveil ont été choisie aléatoirement. Les tâches apériodiques peuvent être préemptées au cours de leurs exécutions et donc d'exécuter en plusieurs fois lors d'une instance.

— **r,r' :** Date aléatoire de réveil
— **C :** Temps d'exécution
— **D :** Délai critique
— **d :** Échéance = r + D

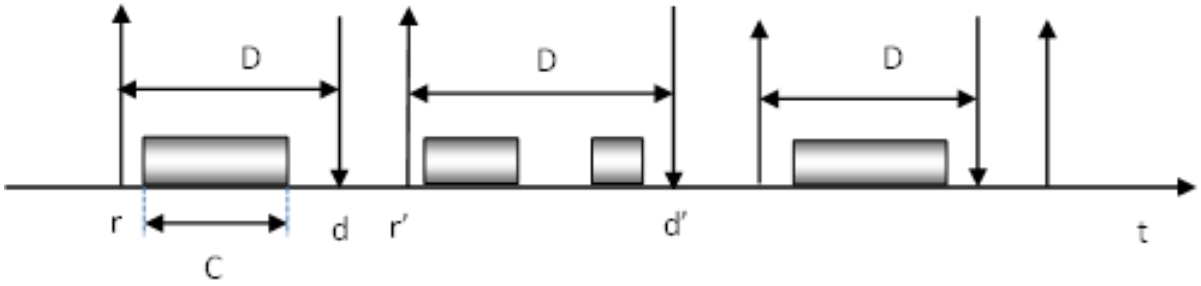

FIGURE 1.10 – Représentation de l'exécution d'une tâche apériodique

1.7.3 Modèle de tâches sporadiques

elles sont activées de façon cyclique à des instants irréguliers. Il existe un intervalle de temps minimal entre deux réveils d'instances successives. Les dates d'activation des différentes instances d'une tâche ne peuvent pas être déterminées a priori [7]. Une tâche sporadique est caractérisée par trois paramètres temporels (C, D, T). Où :

— **C** : Durée d'exécution
— **D** : Délai critique
— **T** : Un intervalle minimal séparant deux occurrences successives (voir Figure 1.11).

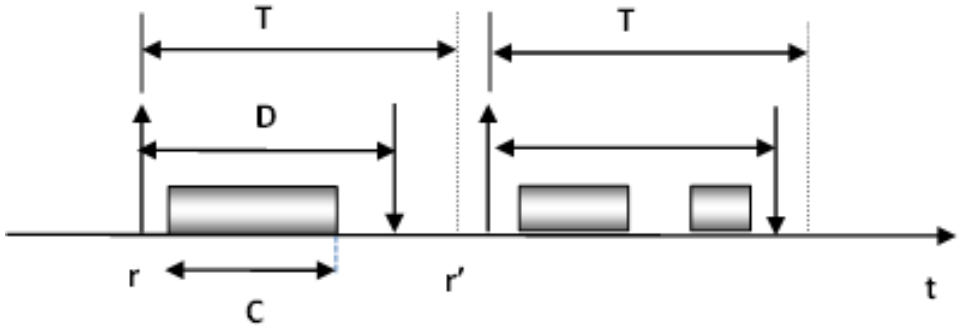

FIGURE 1.11 – Modèle de tâches sporadiques

1.7.4 Modèle de tâches avec contrainte de précédence :

Les tâches qui constituent l'application temps réel, peuvent être liées par des relations de type synchronisation ou de communication qui se traduit par une relation de précédence de transfert de données entre les tâches. Cela implique que certaines tâches ne peuvent s'exécuter que si d'autres tâches se sont exécutées avant.
Sur la Figure 1.12 : la tâche T2 précède la tâche T3 et T4, la tâche T1 précède la tâche T4, la tache T3 et T4 précèdent la tâche T5.

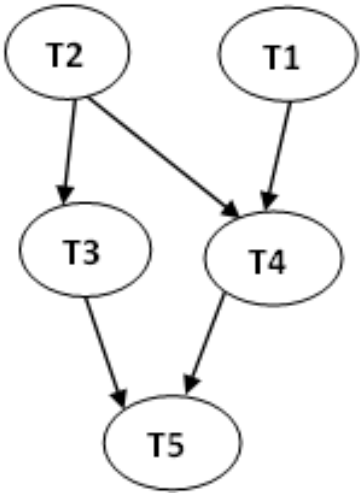

FIGURE 1.12 – Traduction de la relation de précédence d'exécution entre les tâches par un graphe de précédence

1.7.5 Modèle de tâches avec contrainte de ressources critiques :

Les tâches qui utilisent des éléments mis en commun au niveau du système. Certains de ces ressources comme les zones mémoires, ne sont pas ou ne doivent pas être accessibles, par plus d'une tâche à la fois, elles sont dites **ressources critiques**[2]. Une tâche T_i de durée totale C_i qui utilise une ressource critique R possède dans son code une zone ,protégée, appelée section critique, pendant laquelle elle accède à cette ressource. Cette section critique est protégée par des primitives permettant de gérer l'exclusion mutuelle comme un sémaphore par exemple. Par conséquent, en terme de temps, l'exécution de cette tâche peut être décrite par :
$\alpha_{i,k}$ temps avant la section critique.
$\beta_{i,k}$ durée de la section critique (ressource utilisée).
$\gamma_{i,k}$ temps après la section critique.

Ces trois valeurs doivent satisfaire à l'égalité suivante : $C_i = \alpha_{i,k} + \beta_{i,k} + \gamma_{i,k}$ (voir Figure 1.13)

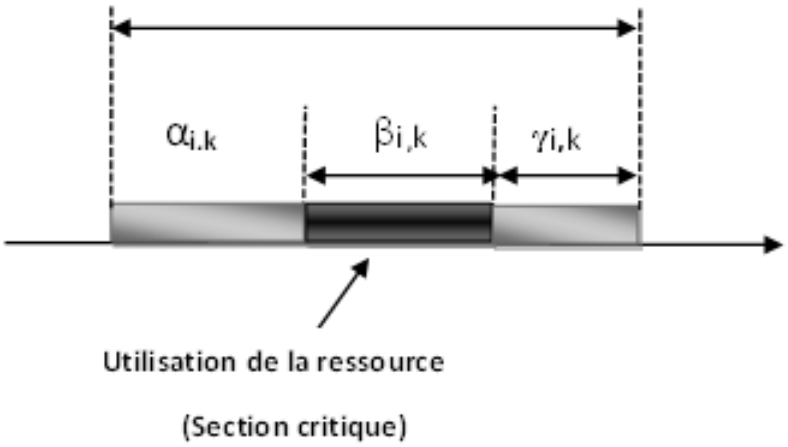

FIGURE 1.13 – Représentation d'une tâche contenant une section critique

1.8 Ordonnancement temps réel :

Dans les systèmes temps réel, l'étude de l'ordonnancement et l'affectation de priorités est primordiale pour assurer un fonctionnement sûr du système. Le système temps réel

multitâches permet l'exécution de plusieurs tâches à la fois. Lorsque plusieurs tâches demandent à s'exécuter simultanément sur un même processeur, plusieurs problèmes se posent. Pour permettre le respect des échéances des tâches, il faut prévoir un mécanisme d'ordonnancement permettant le bon fonctionnement du système.

1.8.1 Ordonnancement :

est de définir une stratégie ou une méthode permettant l'affectation des tâches au processeur, afin d'assurer le respect de toutes les contraintes temporelles en fonctionnement normal et le respect des contraintes temporelles des tâches les plus critiques en fonctionnement anormal.

1.8.2 Algorithme d'ordonnancement

Un algorithme d'ordonnancement est un algorithme capable de donner une description (séquence) du travail à effectuer par le ou les processeurs, une séquence est dite **valide** si les échéances des tâches sont respectées[3]. Un algorithme est dit **fiable** pour une configuration de tâches s'il produit une séquence valide sur une durée infinie quelles que soient les valeurs des premières dates de déclenchement des différentes tâches. Une configuration est dite **ordonnançable** s'il existe au moins un algorithme fiable[3].

Dans un contexte de tâches et d'algorithmes d'ordonnancement (affectation de priorités), nous allons qualifier l'algorithme d'ordonnancement étudié selon deux aspects[3] :

optimalité : Si la configuration de tâches est ordonnançable dans cette catégorie d'algorithmes, alors elle le sera avec l'algorithme étudié ;

ordonnançabilité : La capacité à pouvoir prévoir l'ordonnancement de la configuration de tâches en se basant sur des conditions nécessaires et/ou suffisantes ou des simulations de l'exécution.

1.8.3 Catégorisation des algorithmes d'ordonnancement :

Les algorithmes d'ordonnancement sont classifiés selon les caractéristiques du système sur lequel ils sont implantés. On indique ci-dessous quelques caractéristiques

1.8.3.1 Monoprocesseur ou multiprocesseur

Si toutes les tâches ne peuvent s'exécuter que sur un seul et même processeur l'ordonnancement est de type monoprocesseur. Et si plusieurs processeurs sont disponibles dans le système, l'ordonnancement est de type multiprocesseur.

1.8.3.2 Hors-ligne ou en-ligne

Les algorithmes d'ordonnancement peuvent être classés en deux catégories :

Un ordonnancement hors-ligne la séquence d'ordonnancement est établi avant le lancement de l'application et est ensuite implémentée dans une table au niveau du séquenceur et exécutée en ligne par le processeur, ce type de l'ordonnancement n'est pas possible que lorsque l'on connait à l'avance le moment où les tâches seront prêtes.

Un ordonnanceur en-ligne la séquence d'exécution se construit dynamiquement en fonction des événements (signal de réveil périodique d'une tâche, libération d'une ressource critique...) qui surviennent et peuvent modifier à tout moment la liste des tâches prêtes. Quelle que soit la méthode utilisée pour élaborer la stratégie en-ligne, elle se doit d'être peu consommatrice de ressources. En effet, le temps consacré par les ressources de calcul à l'ordonnanceur pour exécuter sa stratégie s'appelle le surcoût processeur (livre ordonnancement)

1.8.3.3 Ordonnancement à priorité fixe vs dynamique :

La majorité des algorithmes d'ordonnancement en-ligne ordonnancent les tâches prêtes en leur associant une valeur appelée priorité. À chaque décision d'ordonnancement, le processeur est attribué à la tâche prête la plus prioritaire. Lorsque les priorités sont attribuées au démarrage de l'application et pour toute sa durée de vie, nous parlons alors d'ordonnancement à priorité fixe. Ainsi pour une tâche périodique, toutes ses instances auront la même priorité. Si les priorités peuvent changer durant l'exécution de l'application, nous parlons d'ordonnancement à priorité dynamique.

1.9 Algorithmes d'ordonnancement temps réel :

Plusieurs algorithmes d'ordonnancement temps réel ont été proposés. Ces algorithmes sont classés selon la somme des informations qu'ils requièrent pour effectuer l'ordonnancement des tâches. Dans cette section on présent ces algorithmes d'ordonnancement en précisant les contraintes qui sont prises en compte ainsi que les conditions d'ordonnançabilité si celles-ci existent.

1.9.1 Algorithmes d'ordonnancement de tâches indépendantes périodiques :

1.9.1.1 Algorithmes à priorités fixes aux tâches :

Il s'agit de l'affectation des priorités fixes aux tâches, avant la mise en fonctionnement du système, ces algorithmes sont généralement basés sur des contraintes temporelles statiques des tâches comme la période ou le délai critique. On présente ici les deux principaux algorithmes d'ordonnancement à priorités fixe, algorithme RM et DM.

Algorithme RM (Rate Monotonic) : Basé sur une règle d'affectation des priorités aux tâches selon la période c-à-d. plus la période de la tâche est petite, plus la priorité de la tâche est grande. La tâche conserve cette priorité pendant toute son exécution. L'algorithme RM est optimal dans la classe des algorithmes à priorités fixes ; pour des systèmes de tâches indépendantes, synchrones, et périodiques à échéances sur requête ($D_i = P_i$). Une condition suffisante d'ordonnançabilité d'une configuration de n tâches périodiques est obtenue pour un facteur d'utilisation U du processeur, si U satisfait l'intégralité suivante :

$$U = \sum_{i=1}^{n} \frac{Ci}{Pi} \leq n(2^{\frac{1}{n}} - 1) \tag{1.4}$$

n nombre de tâches.

On supposant que on a le tableau suivant qui donne les valeurs de U pour un nombre de tâche de 1 à 10

Cette équation exprime que lorsque n$\rightarrow \infty$, l'utilisation du processeur doit rester inférieur à 69.3%, c'est une condition suffisante (mais non nécessaire).

On peut faire deux remarques concernant l'algorithme RM :

1. Une configuration de n tâches est ordonnançable si le facteur d'utilisation U (occupation

N	U
1	100
2	82,8
3	78
4	75
...	...
10	71
∞	69

TABLE 1.1 – Exemple de facture d'utilisation des tâches

du processeur) ne dépasse pas 69% .

2. Une configuration de n tâches peut être ordonnançable ou non avec cette affectation de priorités selon RM si le facteur d'utilisation U dépasse 69%.

Exemple : Tâche A (r0= 0, C=2, D=6, P=6)

Tâche B (r0=0, C=3, D=5, P=8)

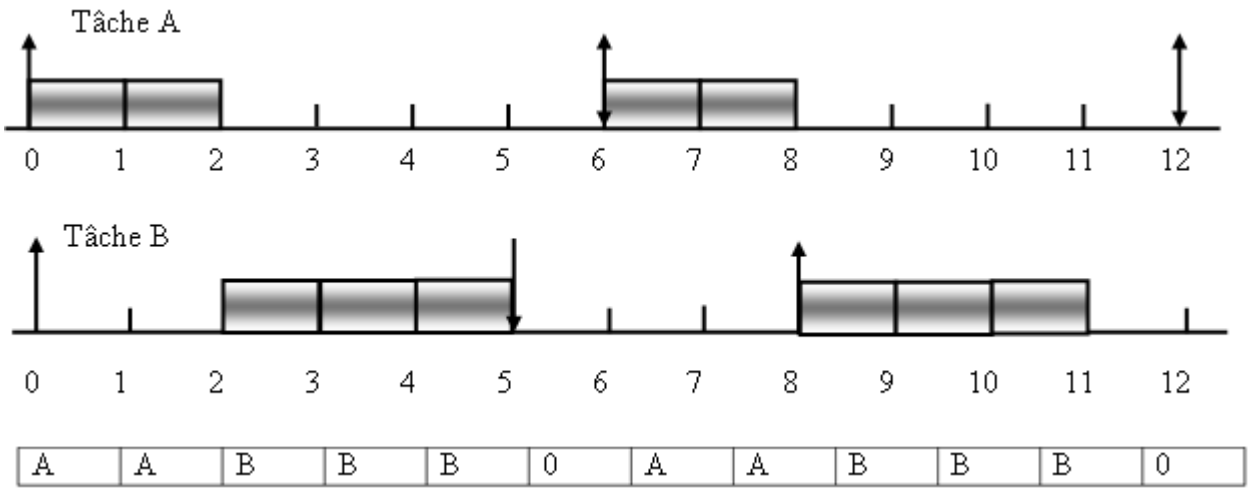

FIGURE 1.14 – Exemple d'ordonnancement RM

Algorithme DM (Deadline Monotonic) : Appelé aussi (Inverse Deadline (ID)), il s'agit d'affecter la priorité la plus grande à la tâche dont le délai critique est le plus petit. L'algorithme DM est optimal dans la classe des algorithmes à priorités fixes ; pour des systèmes de tâches indépendantes, synchrones, et à échéances contraintes (échéances inférieures aux périodes $D_i < P_i$). Une condition suffisante d'ordonnançabilité d'une configuration de n tâches périodiques est obtenue pour un facteur d'utilisation U du processeur, si la condition suivante est vérifiée :

$$U = \sum_{i=1}^{n} \frac{Ci}{Di} \leq n(2^{\frac{1}{n}} - 1) \tag{1.5}$$

n nombre de tâches

Exemple : Tâche A (r0= 0, C=2, D=6, P=6)
Tâche B (r0=0, C=3, D=5, P=8)

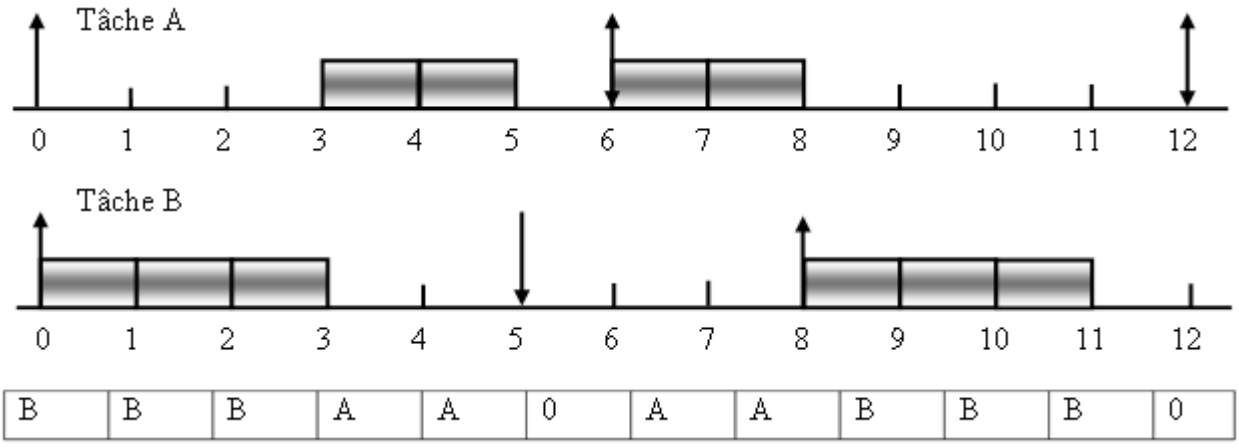

FIGURE 1.15 – Exemple d'ordonnancement DM

1.9.1.2 Algorithmes à priorités dynamiques :

ces algorithmes sont généralement basés sur des contraintes temporelles variables comme l'échéance la plus proche. Ces algorithmes doivent être exécutés, dans les meilleurs cas, à chaque instant de réveil afin d'actualiser les priorités des différentes tâches. On présente ici les deux principaux algorithmes d'ordonnancement à priorités dynamiques, Earliest Deadline et Least Laxity.

Algorithme EDF (Earliest Deadline First) : Appelé aussi (Relative Urgency) est un algorithme d'ordonnancement dynamique. Avec EDF, une tâche est d'autant plus prioritaire que son urgence est forte c'est-à-dire que sa date d'échéance est la plus proche de la date courante. L'algorithme EDF est optimal pour les ordonnancements à priorité dynamique si les échéances sont inférieures aux périodes ($D_i < P_i$), mais connues au réveil. Les priorités sont dynamiquement attribuées en fonction des échéances, au fil du temps. Afin qu'un ensemble de n tâches soit ordonnançable pour EDF, il suffit que

$$U = \sum_{i=1}^{n} \frac{Ci}{Di} \leq 1 \tag{1.6}$$

La condition est nécessaire et suffisante si $\forall i$, Di = Pi

Exemple : Tâche A (r0= 0, C=2, D=4, P=6)
Tâche B (r0=0, C=3, D=8, P=8)
Tâche C (r0=0, C=1, D=3, P=4)

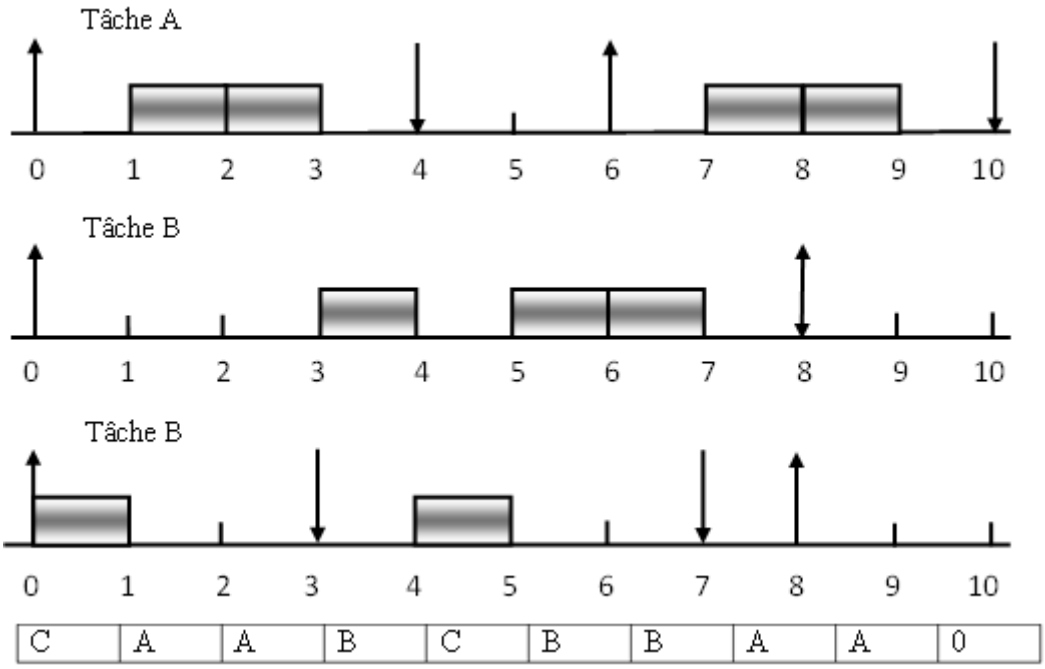

FIGURE 1.16 – Exemple d'ordonnancement EDF

Algorithme LLF (Least Laxity First) : Cet algorithme s'appuie sur un paramètre temporel dynamique dit Laxité. La laxité est définie par l'écart maximal entre la date

d'activation de la tâche et sa date de démarrage de sorte que l'échéance soit respectée. LLF attribue à tout instant (variable) la plus haute priorité à la tâche ayant la plus petite laxité. L'algorithme LLF est optimal dans la classe des algorithmes d'ordonnancement en-ligne, pour des systèmes de tâches indépendantes et à échéances inférieures ou égales aux périodes. Cet algorithme LLF a les mêmes caractéristiques que l'ordonnancement EDF [4] : optimalité et ordonnançabilité.

Exemple :

Tâche A (r0=0, C=3, D=7, P=20)

Tâche B (r0=0, C=2, D=4, P=5)

Tâche C (r0=0, C=1, D=8, P=10)

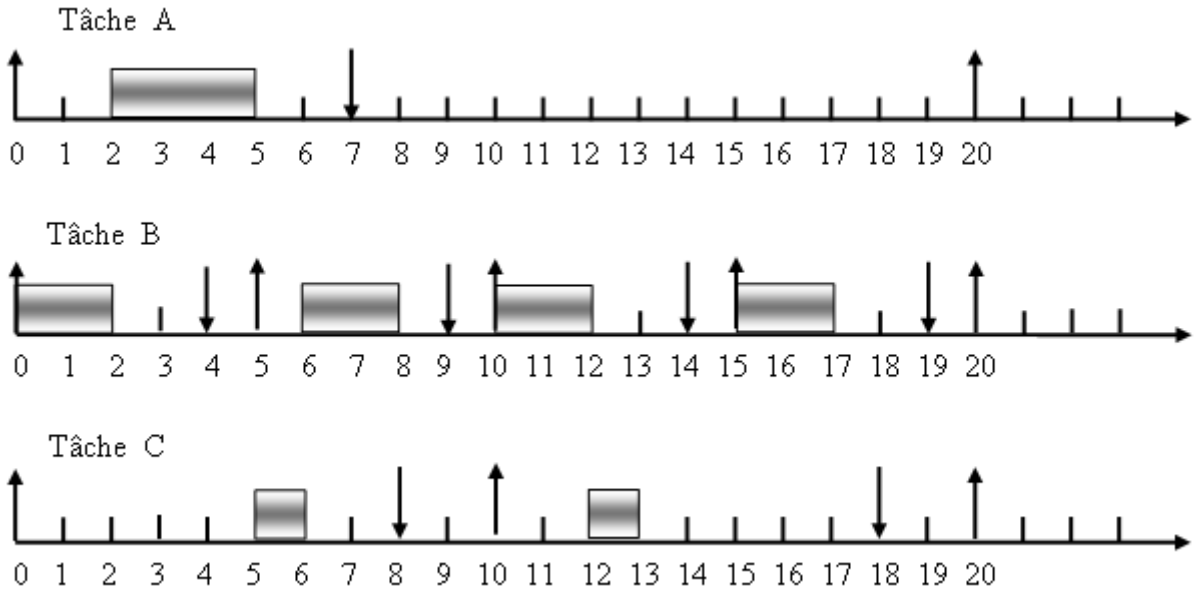

FIGURE 1.17 – Exemple d'ordonnancement LLF

1.9.2 Ordonnancement des tâches indépendantes apériodiques :

Certains systèmes temps réel doivent prendre en compte l'arrivée des tâches apériodiques, celles-ci doivent être intégrées dans un ordonnancement des tâches périodiques et ce, en respectant les contraintes temporelles de ces dernières. Plusieurs approches ont été proposées pour l'ordonnancement des tâches apériodiques.

1.9.2.1 Approche basée sur un traitement en arrière-plan :

La méthode d'ordonnancement en arrière-plan est simple. Les tâches apériodiques sont ordonnancées selon un ordre FIFO lorsque le processeur est inactif et leurs temps de réponse croient avec la charge périodique. Donc, les temps de réponse des tâches apériodiques peuvent être mauvais surtout si la charge des tâches périodiques est importante

Exemple :

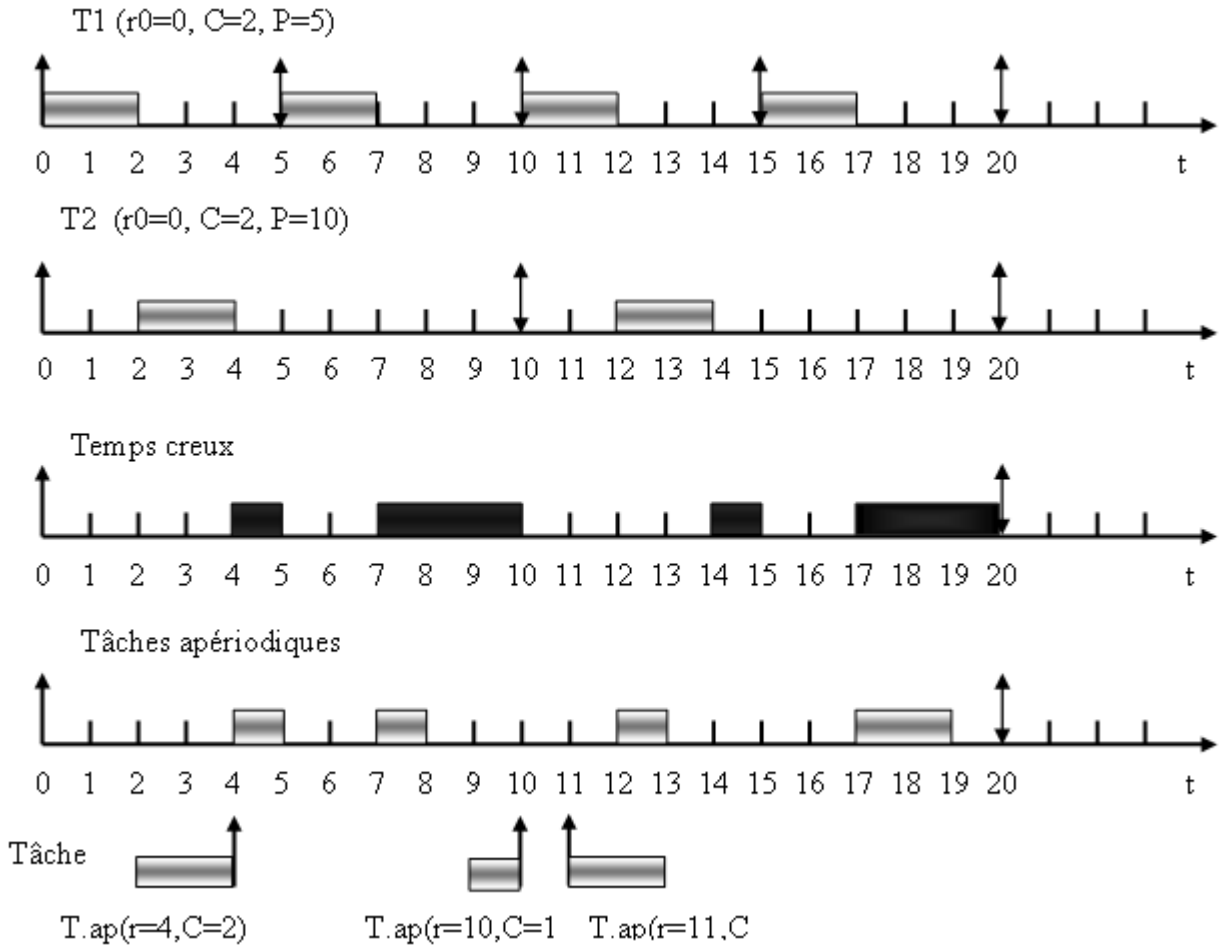

FIGURE 1.18 – Ordonnancement en arrière-plan

1.9.2.2 Approche basée sur les serveurs de tâches :

un serveur est une tâche périodique créée spécialement pour veiller à l'ordonnancement des tâches apériodiques. Elle est caractérisée par une période et un temps d'exécution

(la capacité du serveur). Elle est souvent ordonnancée avec le même algorithme que les autres tâches périodiques. Une fois active, la tâche serveur sert les tâches apériodiques dans la limite de sa capacité. L'ordre de service des tâches apériodiques ne dépend pas de l'algorithme d'ordonnancement des tâches périodiques (il peut être FIFO, f(échéances), f(temps d'exécution), ...)[8].

Il existe plusieurs types de serveurs : le plus simple est le serveur par scrutation. Les autres (serveur ajournable, à échange de priorité, sporadique) en sont des améliorations.

Ordonnancement avec un serveur par scrutation PS (Polling Server) : Le premier type de serveur le plus simple, à chaque activation, le serveur traite périodiquement les tâches apériodiques en attente selon la technique FIFO depuis son activation précédente, jusqu'à épuisement de sa capacité (ou plus de tâches en attente).
Si, lors d'une nouvelle activation, il n'y a aucune tâche apériodique en attente, le serveur se suspend jusqu'à la prochaine occurrence : sa capacité (temps d'exécution) est récupérée par les tâches périodiques[8]. Ainsi toutes les tâches périodiques y compris la tâche serveur sont ordonnancées selon l'algorithme RM. Le PS permet de minimiser le temps de réponse des tâches apériodiques en les assimilant à une tâche périodique sans compromettre l'exécution des tâches périodiques[5].
Exemple (voir Figure 1.19) :
Deux taches périodiques T1 et T2

1.9.3 Ordonnancement des tâches périodiques dépendantes :

Une dépendance entre les tâches peut être de deux types : une dépendance avec contraintes de précédences et /ou une dépendance avec contraintes de ressources.

1.9.3.1 Ordonnancement avec contraintes de précédences :

Des tâches peuvent être liées par des contraintes de précédences lorsqu'elles ont des relations de synchronisation (sémaphores, évènement) ou de communication. On appelle une contrainte de précédence entre la tâche T_i et la tâche T_j le cas où T_i précède T_j, si T_j doit attendre la fin d'exécution de T_i pour commencer sa propre exécution.

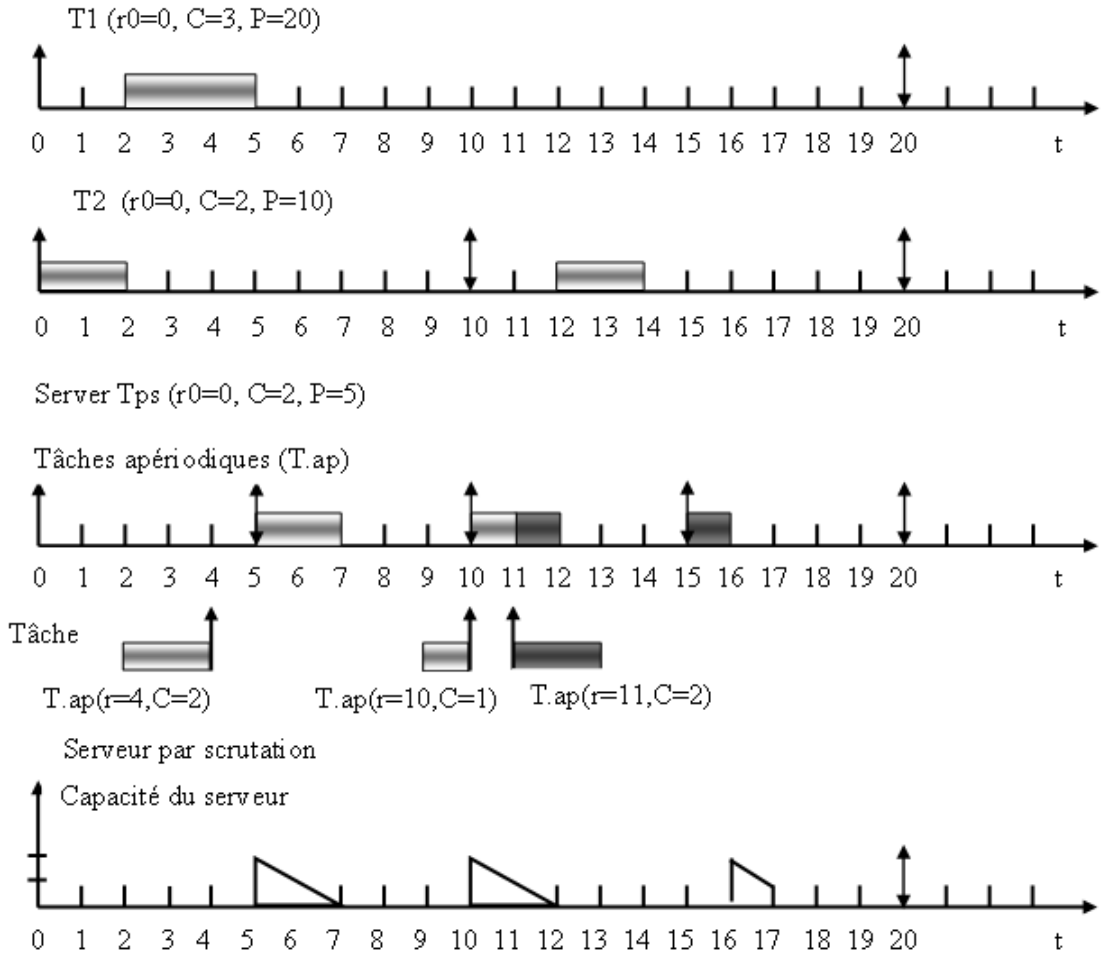

FIGURE 1.19 – Ordonnancement avec un serveur par scrutation.

Prise en compte des relations de précédence avec RM : La politique d'ordonnancement RM affecte des priorités fixes aux tâches de plus courte période . Lorsque les tâches sont périodiques et de même période, l'algorithme RM leur affecte des priorités de manière arbitraire. La prise en compte des relations de précédence dans ce cas se traduit par :

$r*_i$=Max$\{r_i, (r*_{prédécesseur})\}$ pour $T_{prédécesseur}$ tâche précédant T_i

Si A précède B avec PA=PB alors Priorité(A)>Priorité(B) selon RM

Exemple Soit le graphe de précédence des tâches de la Figure 1.20) tel que A précède B et C et D précèdent E. A et B sont de période 5 ; C, D et E sont de période 6. Les priorités des tâches doivent respecter les règles de précédence et celles de RM comme dans le Tableau 1.2).

Tâche	**A**	**B**	**C**	**D**	**E**
Priorité	5	4	2	3	1

TABLE 1.2 – Exemple relations de précédence avec RM

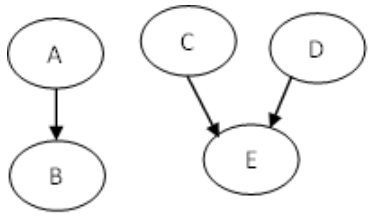

FIGURE 1.20 – Exemple Graphe de précédence selon RM

Prise en compte des relations de précédence avec EDF La modification des contraintes doit permettre, lors de l'ordonnancement, vérifiera les relations de précédence c'est-à-dire, dans le graphe de précédence de commencer les tâches après leurs prédécesseurs. Les échéances sont alors modifiées de manière à associer à une tâche une échéance di ($d_i = r_i + D_i$) strictement inférieure à celles de ses successeurs [9] étant donné que le calcul de priorité est basé sur cette échéance. Ces modifications sont comme suit :

$r*_i$=Max(r_i, ($r*_{prédécesseur} + C_{prédécesseur}$))

$d*_i = Min(d_i, (d*_{successeur} - C_{successeur}))$

1.9.3.2 Ordonnancement des tâches avec contraintes de ressources :

La majorité des applications temps réel utilisent des ressources critiques. Il est donc nécessaire d'intégrer le partage de ressources aux algorithmes d'ordonnancement. On distingue deux types de ressources : les ressources matérielles (ex : la mémoire) et les ressources logicielles (tel que les variables). On dit que ressource est critique lorsqu'elle n'accepte qu'un seul accès à la fois. L'ordonnancement d'un système de tâches en présence de ressources critiques peut produire deux phénomènes ou problèmes qui sont :

- **L'inversion de priorité :** ce problème survient lorsqu'une tâche de faible priorité bloque une tâche de forte priorité (voir Figure 1.21).

- **L'inter blocage :** ce phénomène se produit lorsque deux ou plusieurs tâches sont bloquées car chacune a demandé l'accès à une ressource en possession de l'autre (voir

Figure 1.22).

Dans le but d'éviter le phénomène de l'inversion de priorité et le problème d'inter blocage, plusieurs protocoles de gestion de ressources critiques ont été proposés.

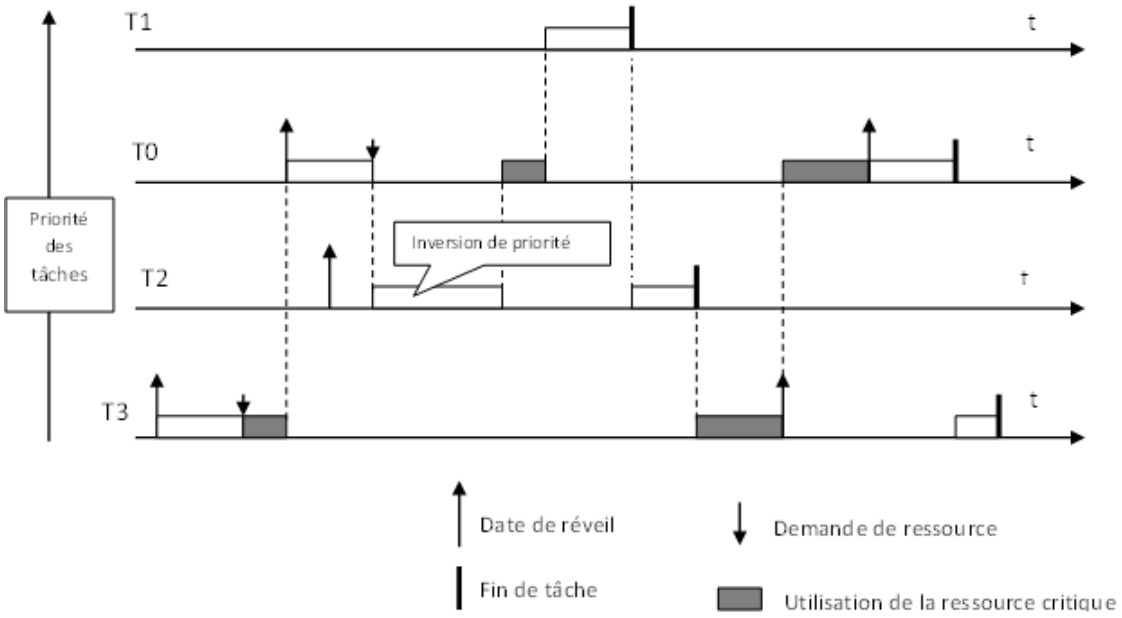

FIGURE 1.21 – Représentation du phénomène d'inversion de priorité

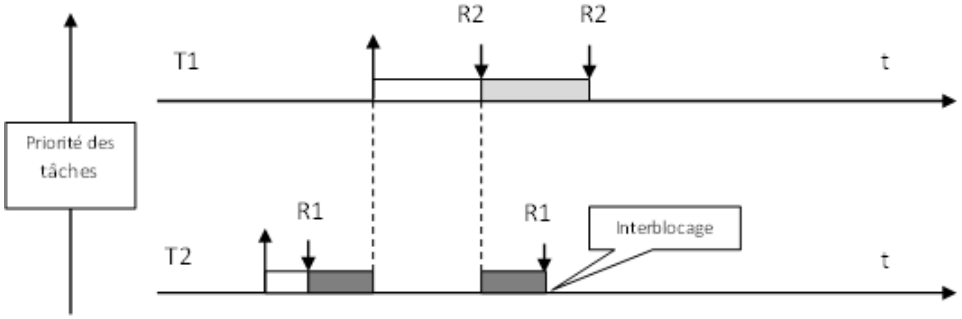

FIGURE 1.22 – Représentation du phénomène d'interblocage

Protocole à héritage de priorité : ce protocole vise à résoudre le problème de l'inversion de priorité. Le principe consiste à interdire toute préemption de la tâche possédant une ressource demandée par des tâches plus prioritaires (tâches bloquées). Pour cela on monte la priorité de la tâche bloquante au niveau de celle bloquée, une fois la ressource

libérée, la tâche bloquée reprend sa priorité initiale. Notons que ce protocole n'est utilisé qu'avec des algorithmes d'ordonnancement à priorité fixes. Ainsi, le problème de l'inter blocage reste possible(voir Figure 1.23).

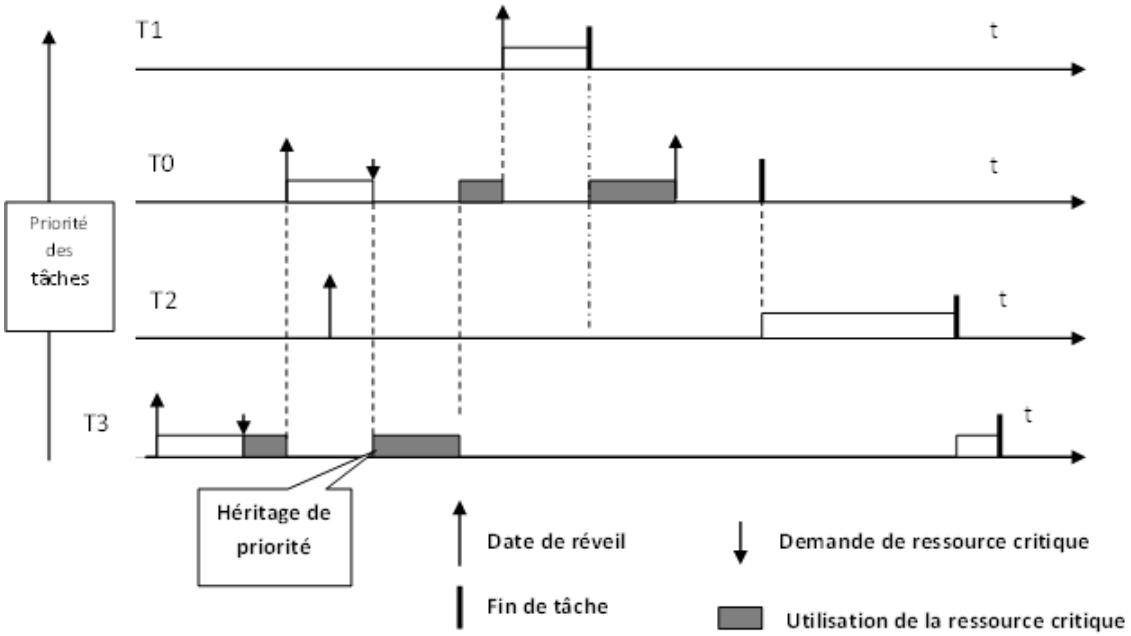

FIGURE 1.23 – Evitement de l'inversion de priorité grâce au protocole d'héritage de priorité

Protocole de priorité plafond : ce protocole permet de répondre au problème de l'inversion de priorité et celui de l'inter blocage, donc a été conçu pour résoudre les limitations de protocole à héritage de priorité ; le principe consiste à affecter à chaque ressource une priorité plafond, qui est la plus haute des tâches qui y accèdent. Le mécanisme du protocole de traitement des ressources est le suivant [2] :

- Si la ressource est libre :
  - Une tâche accède à cette ressource si sa priorité est strictement supérieure au plafond système ou sinon est responsable de la valeur actuelle du plafond.
  - Sinon la tâche est bloquée et la tâche possédant la ressource hérite de la priorité de la tâche bloquée.
- Si la ressource n'est pas libre : la tâche est bloquée et la tâche possédant la ressource hérite de la priorité de la tâche bloquée.

Notons que ce protocole est destiné aux algorithmes d'ordonnancement à priorité fixes. L'avantage de ce protocole est d'empêcher tout inter blocage et d'améliorer la limitation du nombre de blocages. L'inconvénient majeur de ce protocole est la complexité de la mise en œuvre.

Une tâche T_i ne peut être bloquée que pendant la durée d'une section critique d'une tâche de priorité inférieure utilisant une ressource de priorité plafond plus grande ou égale à celle de T_i.

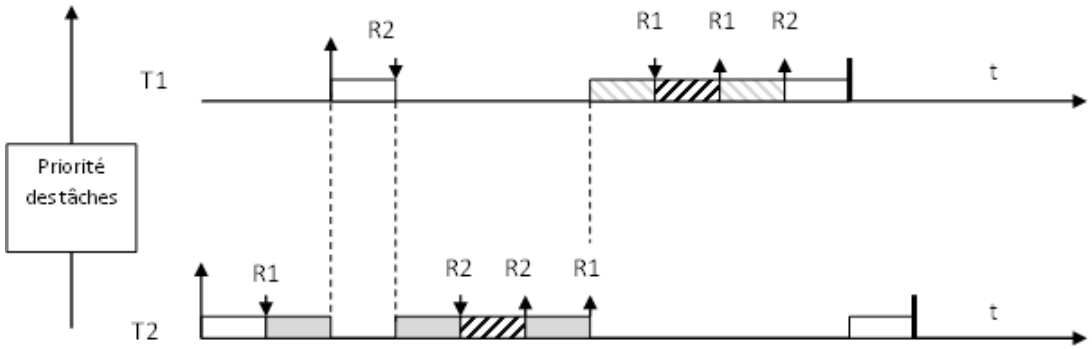

FIGURE 1.24 – Evitement de l'interblocage grâce au protocole à priorité plafond

Protocole d'allocation de la pile : ce protocole est une amélioration du protocole à priorité plafond. Il est adapté à l'algorithme d'ordonnancement EDF. Consiste à définir une nouvelle notion appelée niveau de préemption d'une tâche qui correspond à une seconde priorité qui doit être fixe et qui sert à définir la condition d'autorisation de préemption. Une tâche T_j ne peut préempter une tâche T_i que si les conditions suivantes sont vérifiées :

- T_j est plus prioritaire que la tâche T_i (ça dépend de l'algorithme d'ordonnancement).
- Le niveau de préemption de T_j est supérieur à celui de T_i.
- Le niveau de préemption de T_j est supérieur au plafond système (la plut forte valeur plafond parmi celle des ressources).

En utilisant ce protocole, une tâche n'est pas autorisée à démarrer son exécution tant que toutes les ressources qui lui sont nécessaires ne sont pas disponibles, ce qui permet

d'éviter l'interblocage. Le PAP limite le nombre de sections critiques pouvant bloquer une tâche à un.

1.10 Conclusion :

Dans ce chapitre nous avons présenté, les notions de base nécessaires à la compréhension du principe de l'ordonnancement temps réel et nous nous sommes intéressés à un système temps réel caractérisé par une architecture monoprocesseur.

Ce chapitre regroupe les généralités sur le système informatique temps réel, dont nous avons donné quelques définitions sur ces systèmes, les caractéristiques, l'architecture, modélisation des tâches et les algorithmes d'ordonnancement.
L'ordonnancement temps réel régulé sera traité dans le chapitre suivant.

Chapitre 2

Ordonnancement temps réel régulé

2.1 Introduction

La plupart des machines inventées par l'homme nécessitent un système de régulation ou de contrôle pour fonctionner. Ces systèmes de contrôle existaient avant l'invention des ordinateurs. Les systèmes de contrôle des procédés permettent d'assurer un haut degré d'efficacité et d'offrir une multitude d'avantages, comme l'automatisation de tous les processus, donc réduire l'intervention humaine. On dit que le processus est contrôlé, piloté ou supervisé par le système qui réagit aux changements d'état du processus. Le principe général de commande des procédés est connu sous le terme de commande en boucle fermée ou commande par rétroaction.

2.2 Limites d'ordonnancement temps réel classique :

1- Connaissance des paramètres temporels : Les algorithmes d'ordonnancement temps réel classique (RM, DM...), considèrent que l'on connaisse à l'avance la valeur des paramètres temporels des tâches. Si toutes les tâches ne sont pas périodiques, le problèmes peut être de plus ou moins bien contourné par une "mise en réserve " de temps CPU, ce qui suppose tout de même de connaitre une borne inferieure du temps séparant deux activations successives[10].

La détermination de la durée d'exécution des tâches est un problème majeur (voir la section 2.6). Cette durée dépend de plusieurs facteurs, comme le langage hôte, la machine

cible et du contexte. Les branchements conditionnels dans le code génèrent aussi des durées des exécutions variables.

L'incertitude sur la durée d'exécution des tâches peut provenir aussi de l'algorithme lui-même. Cependant, une connaissance de la durée d'exécution au pire cas ne garantit pas forcément l'obtention d'un ordonnancement fiable.

2- Politique d'affectation des priorités : Les politiques d'affectation des priorités vues dans les algorithmes (RM, DM, EDF ...) présentent un certain caractère d'optimalité, e.g. utilisation maximale du CPU, donc du point de vue de l'informatique et en dehors de tout contexte applicatif. Or, une application de commande ordonnancée "en aveugle" selon une de ces méthodes peut présenter un comportement médiocre, alors qu'un découpage de l'algorithme et une affectation de priorité suivant des notions d'urgence du d'importance relatives, dépendant du contexte applicatif, peut conduire à obtenir des performances de l'application (temps de réponse, stabilité, faible erreur de poursuite...) beaucoup plus satisfaisantes[10].

3- Environnement dynamique : Dans la littérature classique concernant l'ordonnancement temps réel, les algorithmes sont étudiés séparément du contexte applicatif. Cette démarche peut conduire à préconiser un ordonnancement "optimal" du point de vue par exemple de l'utilisation de la ressource de calcul mais inefficace du point de vue du contrôle de l'application qui est tout de même le point de départ[11].

2.3 Stratégies de commande des procédés

Le système temps réel est associé dans des systèmes de commande dans le but de contrôler un processus pour l'amener dans un état conforme aux besoins de l'utilisateur. Il peut être utile en régulation, pour maintenir à une valeur constante la sortie du procédé contrôlé, par exemple la vitesse d'un véhicule asservie par un régulateur de vitesse. Il peut être utilisé pour poursuivre un objectif variant dans le temps, exemple l'asservissement de l'orientation d'une antenne radar pour maintenir l'objet observé dans la zone d'activités de l'antenne. Ainsi, pour réaliser cette commande, deux approches de stratégies sont envisageables

— Commande par boucle ouverte ;

— Commande par boucle fermée ;

2.3.1 Commande par boucle ouverte :

Le système de commande n'a pas d'information sur l'évolution et le comportement du système commandé. Dans ce cas, il s'agit d'une régulation sans boucle de contrôle entre l'entrée et la sortie du système, ni retour d'informations. Le système en lui-même n'est pas en mesure de donner la sortie désirée et il ne peut pas prendre en compte les perturbations. Dans ces systèmes, les changements de sortie peuvent être corrigés seulement en changeant l'entrée manuellement (voir Figure 2.2) Ces systèmes sont simples dans la construction, stables et bon marché. Mais ils sont inexacts et peu fiables. De plus, ces systèmes ne tiennent pas compte des perturbations externes qui affectent la sortie et ne déclenchent pas automatiquement des actions correctives, et ils dépendent largement du jugement humain.

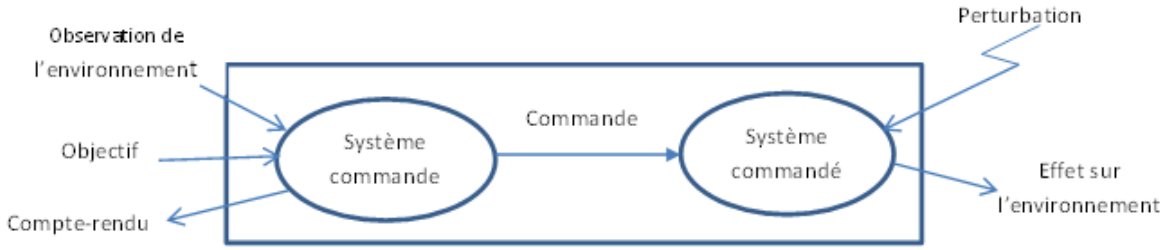

FIGURE 2.1 – Commande par boucle ouverte

Les algorithmes d'ordonnancements traditionnels tels RM et EDF peuvent être considérés comme des ordonnanceurs à boucle ouverte dans la mesure où ils ne font aucun prélèvement sur l'état du système pendant son exécution[1].

2.3.2 Commande par boucle fermée :

La commande en boucle fermée encore dite commande par rétroaction (feedback control, en anglais), le procédé est commandé, de façon continue (répétitive), à partir d'un signal d'erreur résultant de la différence entre la sortie observée (l'état estimé) du procédé et l'état idéal correspondant à l'objectif de commande spécifié [12] Figure 2.2. La

variable commandée (sortie) du système est détectée à chaque instant de temps, en retour et comparée à l'entrée souhaitée.

L'architecture d'un système de commande en boucle fermée comme montré par la figure 2.2 est composée des actionneurs, reçoivent des commandes et permettant d'agir sur son état .Les commandes calculées numériquement doivent être bloquées (le plus souvent en pratique par un bloqueur d'ordre zéro) pour pouvoir agir entre les instants d'échantillonnage .Et aussi des capteurs permettant de l'observer (mesurer) de façon continue le procédé contrôlé. Elles sont échantillonnées pour être transmises au calculateur (contrôleur) numérique[10].

Comparée à l'approche dite en boucle ouverte, laquelle repose uniquement sur la connaissance précise du modèle du procédé et de son environnement, l'approche de commande par rétroaction (ou en boucle fermée) prend en compte les perturbations et effectue l'action corrective. Ces systèmes de contrôle sont précis, stables et moins affectés par le bruit.

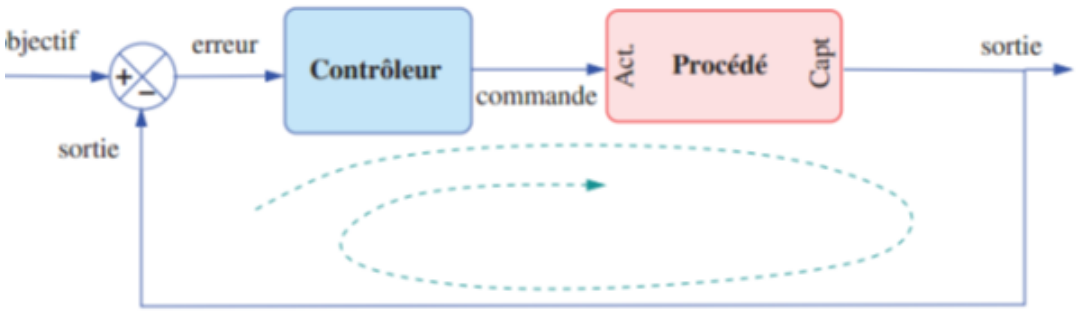

FIGURE 2.2 – Principe de commande par rétroaction

2.4 Paramètres et méthodes d'évaluation des systèmes de commande

2.4.1 Paramètres d'un système de commande

La structure d'un système de commande est représentée sur la figure 2.3 Le système à commander, le capteur et l'actionneur qui est connecté au contrôleur. L'exécution du contrôleur est composée de 3 phases : l'acquisition des mesures, le calcul de la nouvelle commande et la mise à jour de l'actionneur.

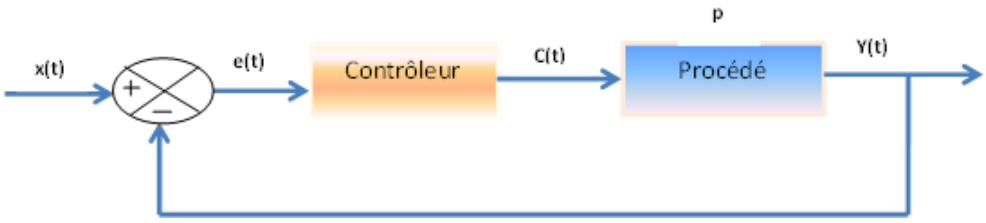

FIGURE 2.3 – La structure d'un système de commande

Les paramètres essentiels du système de commande sont représentés sur la figure 2.3

— X(t) : représente les consignes de l'opérateur au système de contrôle

— Y(t) : Sortie du procédé, est une caractéristique mesurable du procédé.

— E(t) : Erreur ou écart, la différence entre la consigne x(t) et la sortie y(t), e(t)= x(t)-y(t)

— C(t) : commande ou action, c'est un paramètre qui influe sur le comportement du procédé.

— P : Les perturbations sont les autres grandeurs agissant sur le processus ; ce sont des variables aléatoire

2.4.2 Les critères de performances du système de commande

Les critères permettant de qualifier et quantifier les performances du système sont.

2.4.2.1 la stabilité :

le système est stable si à une variation bornée du signal d'entrée correspond une variation bornée du signal de sortie. Une variation d'un signal est dite bornée lorsqu'elle est constante en régime permanent. Le bouclage d'un système peut rendre celui-ci instable (le système est stable s'il ya retour à l'équilibre après disparition de la perturbation, il est instable s'il n'y revient pas ou s'il s'en écarte).

2.4.2.2 la précision :

la précision caractérise l'aptitude d'un système à atteindre la valeur de sortie souhaitée. L'écart entre la consigne (sortie attendue) et la sortie (sortie réelle) se caractérise donc de la manière suivante (entrée et sortie homogènes) : e(t) = x(t) – y(t) L'écart est exprimé

dans l'unité de la grandeur de sortie, ou encore en%.

2.4.2.3 la rapidité :

la rapidité est caractérisée par le temps que met le système à réagir à une brusque variation du signal d'entrée. Cependant, la valeur finale étant le plus souvent atteinte de manière asymptotique (système stable), on retient alors comme principal critère d'évaluation de la rapidité d'un système, le temps de réponse à n%. En pratique, on utilise le temps de réponse à 5% (Tr 5%) appelé aussi temps d'établissement, c'est le temps mis par le système pour atteindre sa valeur de régime permanent à ± 5% près et y rester. C'est une des caractéristiques importantes des systèmes bouclés. Le temps de réponse à 5% caractérise la durée de la phase transitoire. On cherchera souvent à diminuer ce temps de réponse, sans que cela soit au détriment d'autres performances.

2.4.3 Les méthodes d'évaluation des système de commande

Les méthodes d'évaluation des systèmes de commande sont classées en trois catégories :

2.4.3.1 Simulation :

Dans ce cas le système temps réel et les procédés asservis sont simulés par logiciel, par exemple à l'aide de l'outil TrueTime[1].

2.4.3.2 Hardware-in-the-loop :

La simulation Hardware-in-the-Loop (HIL) est une technique utilisée pour le développement et les tests de systèmes de contrôle utilisés pour le fonctionnement de machines et de systèmes complexes. Avec la simulation HIL, la partie physique d'une machine ou d'un système est remplacée par une simulation.

2.4.3.3 Implantation :

L'ensemble est implanté dans une vraie application de régulation embarquée.

La très grande majorité des approches étudiées n'ont été évaluées que par simulation.

2.5 Problématique d'ordonnancement basé sur WCET

Le but des algorithmes d'ordonnancement est de vérifier l'ordonnançabilié de la configuration des tâches en assurant que toutes respecteront leurs échéances temporelle. Cette vérification repose sur la connaissance, a priori, des durées d'exécution des tâches. Celui-ci étant variable d'une exécution à l'autre, on utilise comme majorant la pire durée d'exécution (WCET). La distribution du temps d'exécution, voir figure 2.4) pour un processeur embarqué montre qu'une exécution proche du WCET est un événement plutôt rare, ceci provoque un gaspillage de cycles CPU. Cette approche à longtemps été utilisée pour les configurations de tâches de petite taille s'exécutant sur des processeurs déterministes avec une faible variation des durées d'exécution des tâches temps réel. Cependant les nouvelles générations de processeurs ont une architecture peu déterministe, il devient difficile de prédire des durées d'exécution et la détermination précise des WCET[13]

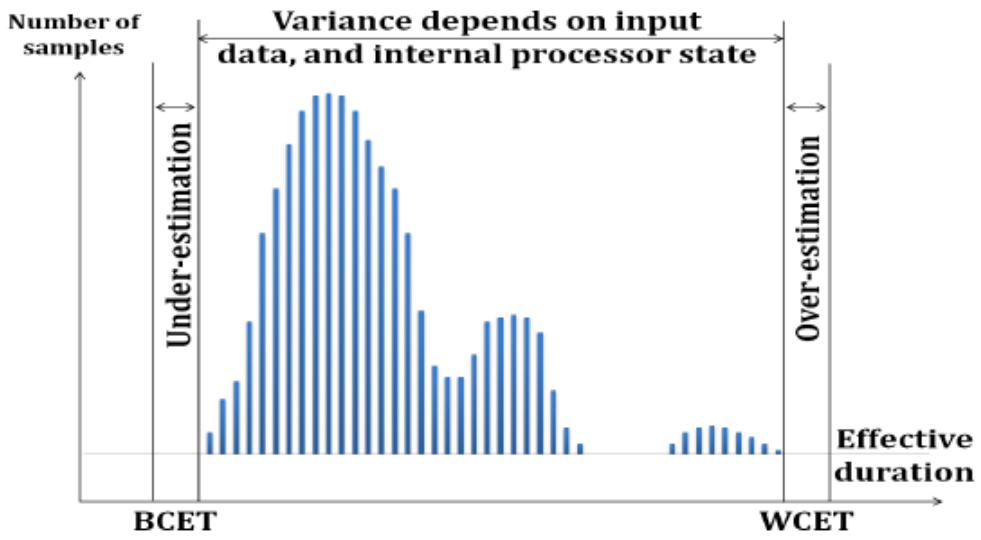

FIGURE 2.4 – La distribution du temps d'exécution[1]

2.6 Ordonnancement temps réel régulé

L'ordonnancement temps réel régulé (real-time feedback scheduling) consiste à appliquer le principe de régulation à boucle fermée (rétroaction) sur des systèmes informatiques dans l'objectif d'améliorer leurs performances pendant leur exécution.

Une caractéristique de l'ordonnancement régulé est qu'il peut être utilisé pour réduire les effets de perturbations et de réduire la sensibilité aux incertitudes. L'idée de la programmation par rétroaction consiste à utiliser le retour d'informations pour maîtriser les incertitudes relatives à la planification des ressources, telles que les variations dans les temps d'exécution des tâches.

La structure générale de la régulation est d'agir sur les paramètres des tâches (période, gestionnaire d'admission, priorité, durée d'exécution de l'algorithme, variante d'algorithme, ...) à partir des mesures de données d'ordonnancement (période, durée d'exécution, temps de réponse, nombre d'échéances dépassées, charge processeur, ...) ou d'informations propres à l'application (erreur de poursuite, niveau de bruit, nombre d'images par seconde, ...)[14]

2.6.1 Les éléments constitutifs d'une boucle de régulation d'ordonnancement temps réel :

Pour construire un régulateur d'ordonnancement à partir d'objectifs spécifiés, il faut définir quelles sont les variables à mesurer, ses sorties (actionneurs) et quelles sont les lois de commande (contrôleurs) utilisables sur le procédé pour agir d'une manière efficace sur celui-ci. Dans ce qui suit on va détailler les éléments qui peuvent être utilisés.

2.6.1.1 Consignes

C'est un signal externe appliqué à un système de contrôle en boucle fermée pour contrôler une action spécifique du procédé. Il représente souvent un comportement idéal de la sortie.

2.6.1.2 Capteurs et mesures :

Les capteurs sont des éléments, transformant la grandeur à mesurer en un signal, représentatif de l'information originelle.
L'état de la ressource d'exécution peut être obtenu au travers de différentes mesures.

— **Charge CPU :** Un des buts principaux d'un régulateur d'ordonnancement est de gérer la charge de calcul d'une ressource d'exécution informatique, il est donc presque toujours nécessaire d'évaluer la charge de travail de la ressource.

— **Le dépassement d'échéances :** sont des évènements faciles à détecter. Ils peuvent cependant être utilisés en complément de l'estimation de charge.
— **La laxité des tâches** (temps restant entre la fin d'exécution d'une instance et l'instant d'activation suivant) peut aussi être envisagée comme indicateur de charge du système.
— Dans le cas d'un système distribué, on devra aussi mesurer ou estimer la charge du réseau, les retards induits, les pertes de meures...
— La quantité de l'énergie disponible dans la batterie

2.6.1.3 Actionneurs :

Les actionneurs sont ici des paramètres d'ordonnancement, agissant sur l'exécution des tâches de plus bas niveau, y compris celles concernant la gestion du processeur. Les variables d'actions disponibles sont :

— Période des tâches : la charge CPU induite par n tâches de durée Ci et de période Pi est :

$$U = \sum_{i=1}^{n} \frac{Ci}{Pi} \tag{2.1}$$

On voit immédiatement que les périodes sont des actionneurs efficaces pour agir sur la charge globale de calcul.
— Priorité des tâches : l'ordre des priorités n'affecte pas la charge de calcul mais l'entrelacement de calcul, et donc les latences mesure / commande. Les priorités doivent aussi refléter l'urgence et l'importance relative des composants sur la performance du contrôleur.
— l'utilisation de variantes (coût d'exécution, performance) d'une même fonctionnalité peut aussi être utilisée comme actionneur sur la charge de calcul.
— Fréquence du processeur : la variation de la fréquence du processeur peut être utilisée comme actionneur, à travers la méthode DVFS, pour modifier la charge du processeur ou réduire sa consommation.

2.6.1.4 Lois de commande :

C'est l'algorithme qui calcule la nouvelle commande en fonction des mesures et des références, est conçue par l'usage d'une des nombreuses méthodes de synthèse de contrôleurs.

Contrôleur PID (Proportionnel-Intégral-Dérivé) : c'est le type de contrôleur le plus utilisé. Il est en général disponible sur la plupart des régulateurs actuellement utilisés. Il permet la stabilisation de la mesure au point de consigne en un temps minimum.

Commande Prédictive : la commande prédictive a joué un rôle très important dans le domaine de contrôle de processus, elle est basée sur l'utilisation d'un modèle pour prédire le comportement futur du système sur un horizon de temps fini. L'objectif du contrôleur prédictif est de maintenir les futures sorties du procédé proches de la valeur souhaitée.

Commande robuste : la commande robuste est une première technique de commande de l'automatique traitant la difficulté d'obtention d'un modèle exact du procédé. [15]Les commandes robustes sont dédiées à la commande de systèmes incertains, elles sont donc particulièrement bien adaptées aux systèmes combinant incertitudes sur le modèle physique du procédé et incertitude sur les paramètres temporels d'exécution.

Commande floue : La commande floue basée sur la théorie des possibilités, qui est proche de la théorie des probabilités. La commande floue est particulièrement adaptée à la commande des systèmes non-linéaires complexes mal modélisés mais pour lesquels une grande expertise humaine existe. La commande floue est parfaitement applicable à des systèmes dont le comportement désiré est décrit de manière plus qualitative que quantitative. Elle peut être appliquée rapidement et donne des bons résultats.

2.7 Travaux réalisés ou algorithmes proposés dans ordonnancement régulé :

Nous présentons ici quelques travaux sur l'ordonnancement régulé. Ces travaux ont pour unique objectif la régulation de paramètres d'ordonnancement, plus exactement la charge processeur ou le nombre d'échéances ratées.

Stankovic et al. (1999) présentent l'algorithme (FC-EDF) qui ajoute une contre réaction à un ordonnancement EDF. Un correcteur PID régule le taux d'échéances ratées d'un ensemble de tâches de durée variable. Pour cela, le correcteur mesure le taux d'échéances

ratées sur une fenêtre de temps et agit sur le temps d'exécution des algorithmes en choisissant des variantes de durée différente. Cette méthode est étendue dans (Lu et al. 2000, 2002) où un second PID, combiné au premier, régule la charge processeur à partir de sa mesure.

Abeni et al. (2002) ajoutent une contre réaction à un serveur de tâches CBS (constant bandwidth server) introduit dans (Abeni and Buttazzo, 1998). Un serveur CBS, assimilable à un processeur virtuel doté d'une portion (bande passante) d'un processeur réel, permet d'y confiner un dépassement d'échéance. La bande passante est normalement fixée à la conception du système à partir de la connaissance des temps d'exécution. Afin de compenser les dépassements d'échéances dus aux variations du temps d'exécution, Abeni et al. (2002) ; Palopoli et al. (2003) utilisent un correcteur PI hybride qui modifie la bande passante en fonction du dépassement d'échéance mesuré.

Buttazzo et al. (1998) ont introduise un modèle de tâche élastique applicable aux tâches périodiques. Chaque tâche est munie d'un coefficient d'élasticité et d'une période minimale et maximale. En réponse à la variation du temps d'exécution des tâches, qui est mesurée, un algorithme heuristique en modifie la période pour attribuer la charge processeur au prorata des coefficients d'élasticité, sous contraintes des périodes admissibles.

Cervin et al. (2002) ont proposé une régulation de l'ordonnancement de plusieurs lois de commande. La période des lois de commande est modifiée en réponse à la variation de la charge du processeur. La conception du régulateur repose sur le travail de (Eker et al. 2000)[16] ayant formulé un problème d'optimisation dont l'objectif est de trouver les périodes optimales d'un ensemble de commande, chacune munie d'une fonction de coût dépendant de sa période, sous contrainte d'une charge processeur maximale.

Cervin (2003) [17] Propose un mécanisme d'ordonnancement de rétroaction, ce mécanisme estime l'utilisation actuelle du processeur. Un filtre passe bas avec un facteur d'oubli noté λ dans l'intervalle [0, 1] est utilisé pour estimer la durée d'exécution sur laquelle on doit se baser pour calculer le facteur d'utilisation du processeur. L'estimation de la durée d'exécution de chaque tâche sur laquelle le régulateur d'ordonnancement peut agir est comme suit :

$$\begin{aligned} \hat{C}_i(0) &= WCET_i \\ \hat{C}_i(k) &= \lambda \hat{C}_i(k-1) + (1-\lambda)c_i \end{aligned} \tag{2.2}$$

où les c_i sont les durées d'exécution réelles des tâches et $\hat{C}_i(0)$ est l'initialisation arbitraire de l'estimateur de chaque tâche pire durée d'exécution. Le paramètre λ permet de lisser l'estimateur dans le temps : s'il est proche de 1, l'estimateur prendra plus de temps pour s'adapter aux variations du temps d'exécution de la tâche, par contre s'il est faible, il sera très sensible à des changements transitoires. A base des $\hat{C}_i$, l'estimation de l'utilisation du processeur $\hat{U}$ est calculée comme suit :

$$U = \sum_{i=1}^{N} \frac{\hat{C}_i(k)}{P_i} \tag{2.3}$$

Si $\hat{U} < U_{ref}$, toutes les tâches peuvent s'exécuter avec leurs périodes nominales. Le terme U_{ref} est la valeur de référence qui représente l'utilisation désirée. Dans le cas contraire, le régulateur d'ordonnancement procède au rééchelonnement des périodes des tâches comme suit

$$P_i = P_{i,nom} \times \frac{\hat{U}}{U_{ref}} \tag{2.4}$$

2.8 Conclusion :

Dans ce chapitre, nous avons conclus que l'objectif principal d'un système de commande est de contrôler un processus pour l'amener dans un état adéquat aux désirs de l'utilisateur.

Le feedback scheduling, consiste à utiliser une boucle de rétroaction pour ajuster l'ordonnancement afin d'optimiser les performances de la régulation et d'améliorer l'utilisation des ressources. Les techniques de commande en boucle fermée peuvent être appliquées à l'ordonnanceur temps réel et le rendant robuste et adaptatif face aux incertitudes temporelles d'exécution.

Dans le chapitre suivant on va traité l'ordonnancement temps réel sous contrainte d'énergie.

Chapitre 3

L'Ordonnancement temps réel sous contrainte d'énergie

3.1 Introduction

L'ordonnancement temps réel sous contrainte de l'énergie s'intéresse à minimiser l'énergie consommée en vue de maximiser la longueur de l'intervalle de temps séparant deux recharges de batterie pour assurer le bon fonctionnement des systèmes. Aujourd'hui nous minimisons massivement la consommation d'énergie avec des technologies innovantes. Celles-ci consistent, soit à diminuer la vitesse de fonctionnement du processeur DVFS (Dynamic Voltage Frequency Scaling), soit à mettre hors tension l'ensemble ou une partie des circuits électroniques DPM (Dynamic Power Management).
Malgré ces améliorations technologiques, tout système embarqué finira par épuiser sa batterie. Pour garantir le fonctionnement de ce système, le remplacement ou la recharge de sa batterie est nécessaire. Cependant, dans certaines systèmes, le remplacement de la batterie est soit coûteux soit impossible. Pour assurer la longévité des batteries, les sources d'énergie alternatives pourraient être exploitées afin de parvenir à une exploitation perpétuelle de ces systèmes : ceci est la récupération d'énergie. Cette approche permet d'étendre la durée de vie des batteries ou les éliminent complètement. Nous notons que l'ordonnancement dans les systèmes de récupération d'énergie constitue un sujet encore très peu étudié actuellement et auquel nous proposons d'apporter notre contribution.

3.2 Système de récupération d'énergie

Un système de récupération d'énergie est un système qui récupère l'énergie ambiante (Energy Harvesting en anglais), cette technique consiste à associer au système consommateur d'énergie, un réservoir assurant le stockage d'énergie récoltée. Un système embarqué (autonome en énergie) se construit autour de trois composantes qui sont (voir figure 3.1)[12] :

- Le récupérateur d'énergie (harvester, en anglais) dont le choix dépend de la nature de l'énergie environnementale, de la quantité d'énergie requise, etc. ;
- Le réservoir d'énergie comme une batterie ou un super-condensateur dont le choix est dicté par les dynamiques du système, des contraintes de dimensionnement ou/et de coût, etc. ;
- Le consommateur d'énergie que représente ici le support d'exécution des tâches temps-réel. Dans ce mémoire, nous considérons que l'énergie consommée par la partie opérative (CPU) du système embarqué.

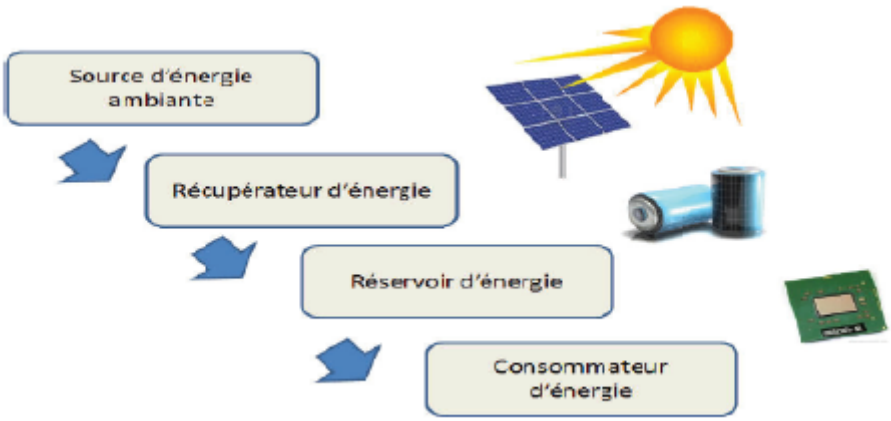

FIGURE 3.1 – Schéma d'un système de récupération d'énergie ambiante

Le système de récupération d'énergie utilise le réservoir d'énergie avec une capacité nominale notée E qui s'exprime en unités d'énergie telle que Joule ou Wattheure. Un réservoir d'énergie (voir la figure 3.2) rechargeable dont le niveau varié entre deux bornes E_{min} et E_{max}, donnant donc une capacité de l'énergie disponible pour le processeur égale à $E = E_{max}$ - E_{min}. Si le réservoir devient plein c'est-à-dire son niveau d'énergie égal à $Emax$ et si on continue à le charger, l'énergie superflue sera alors perdue. Si au contraire le

réservoir d'énergie devient vide c'est-à-dire son niveau d'énergie égal à E_{min}, le processeur devra rester inactif, aucune tâche ne peut s'exécuter.

FIGURE 3.2 – Modèle de batterie rechargeable

3.2.1 Sources d'énergie renouvelable :

Les énergies renouvelables sont des énergies inépuisables. Elles sont issues des éléments naturels : le soleil, le vent, les chutes d'eau, les marées, la chaleur de la terre, la croissance des végétaux ...

3.2.1.1 Énergie solaire :

Le rayonnement solaire constitue l'origine principale des énergies renouvelables. A l'aide de matériaux semi-conducteurs, en particulier le silicium, il devient possible de réaliser des dispositifs, nommés cellules photovoltaïques, qui transforment le rayonnement solaire en électricité et sont donc des convertisseurs d'énergie solaire en énergie électrique. En disposant une cellule photovoltaïque face au soleil, une tension électrique apparait[18].

3.2.1.2 Énergie éolienne :

constitue une des énergies les plus propres. Entrainées par le vent, comme les moulins à vent du passé, les éoliens génèrent des forces mécaniques ou électriques. L'énergie éolienne est devenue un producteur majeur d'énergies renouvelables électriques. L'énergie éolienne est produite par des aérogénérateurs qui captent à travers leurs pales l'énergie cinétique du vent et entrainent elles même un générateur produit de l'électricité d'origine renouvelable[18].

3.2.1.3 Énergie électromagnétique :

L'induction électromagnétique découverte par faraday en 1831, consiste en la génération d'un courant électrique dans un conducteur placé dans un champ magnétique. Dans la plupart des cas, le conducteur est sous la forme d'une bobine et l'électricité est généré par le mouvement d'un aiment dans la bobine grâce à la variation du flux du champ magnétique. Le courant ainsi généré dépend de l'intensité du flux du champ magnétique, de la rapidité du déplacement de l'aimant et du nombre de tours de la bobine.

3.2.1.4 Énergie thermique :

le générateur thermoélectrique est associé à l'effet seebeck découvert par un physicien allemand Thomas Johann Seebeck en 1821. Cet effet se produit lors du passage d'un courant électrique dans un matériau soumis à un flux de chaleur et à un gradient thermique. Ainsi sur ce principe, les thermo générateurs constitués d'un assemblage de jonctions, utilisent l'effet seebeck pour convertir la différence de température entre deux milieux, en électricité.

3.2.2 Les éléments de stockage d'énergie :

L'électricité est une forme d'énergie dite secondaire en ce sens qu'une fois récupérée, elle devient plus difficile à stocker par rapport aux autres formes d'énergie. Afin de pouvoir exploiter au mieux les ressources énergétiques et assurer des pertes minimum, il s'avère incontournable de stocker l'énergie drainée de l'environnement. L'énergie récupérée est stockée dans un élément qui peut être une batterie, pile et/ou supercondansateur.

3.2.2.1 Batteries :

Une batterie est un dispositif électrochimique qui convertit l'énergie chimique en énergie électrique grâce à une réaction chimique d'oxydo-réduction. L'énergie électrique fournie par ces réactions électrochimiques est exprimé par en Watt Heure(Wh). Une batterie est caractérisée par :

— Sa tension, exprimée en Volts(V) ;

— Sa charge électrique en ampère-heure (Ah) ;

— Sa capacité de charge électrique qui représente la charge maximale fournie par batterie ;
— Sa cyclabilité, exprimée en nombre de cycles, caractérise la durée de vie de la batterie ;
— Sa densité d'énergie massique (volumique), exprimée en Watt Heure par Kilogramme, (Wh/kg) ou en Watt Heure par litre (Wh/L) qui représente la quantité d'énergie stockée par unité de masse (volume) de la batterie ;
— Sa densité de puissance massique, exprimée en (W/kg), représente la puissance que la batterie peut délivrer [5].

3.2.2.2 La pile à combustible :

La pile à combustible se définit comme une batterie qui transforme directement l'énergie d'une réaction chimique en énergie électrique de façon continue. Il s'agit d'un générateur d'électricité que d'un stockeur d'électricité, qui de plus possède un très haut rendement énergétique[4].

3.2.2.3 Supercondensateur :

est un condensateur électrochimique qui a une grande capacité de stockage d'énergie. Par rapport à une batterie, le supercondensateur est caractérisé par :

— Charges /décharges très rapides ;
— Langue durée de vie ;
— État de charge vérifiable très facilement ;
— Une tolérance aux basses températures pouvant atteindre les $-40^{\circ}C$ sachant que les batteries ne fonctionnent plus correctement à partir d'une température inférieure à $-10^{\circ}C$.

3.3 L'ordonnancement temps réel et la consommation énergétique du processeur

Dans cette section, nous étudions les différentes technologies de processeurs qui permettent une réduction de la consommation. Nous expliquons, par la suite, quelques notions

portant sur les modèles de consommation d'énergie de ces processeurs. Puis, nous citons les méthodes utilisées pour réduire la consommation d'énergie de ces processeurs.

3.3.1 Technologie des processeurs

La plupart des microprocesseurs disponibles sur les marchés sont conçus dès début d'une faible consommation afin de réduire leur consommation énergétique. Nous distinguons deux classe de processeur en fonction de la possibilité ou non de changer la fréquence nominal de fonctionnement, l'efficacité des stratégies d'ordonnancement, sous contrainte d'énergie, sera naturellement dépendante de cette caractéristique du processeur.

3.3.1.1 Consommation énergétique d'un processeur :

Le circuit CMOS (Complementary Metal Oxide Semiconductor) est la technologie dominante dans les circuits électroniques[19]
L'énergie consommée par un processeur dans un intervalle de temps [a, b] est par définition l'intégrale de la puissance dissipée :

$$E = \int_a^b P(t)\,\mathrm{d}t \tag{3.1}$$

où P(t) est la puissance dissipée à l'instant t. Cette puissance est décomposée en deux types, la puissance statique et dynamique

$$P = P_{dyn} + P_{sta} \tag{3.2}$$

La puissance statique P_{sta} est due à des courants de fuite des transistors et la puissance dynamique P_{dyn} est due à l'activité de commutation :

Dans le circuit CMOS la puissance statique du processeur n'a pas été prise en compte car cette dernière reste négligeable, et donc la puissance s'exprime par :

$$P \approx P_{dyn} \approx cxfxV^2 \tag{3.3}$$

Où c est la somme des capacités dans le circuit chargées et déchargées a chaque cycle, f la fréquence nominale du processeur et V la tension d'alimentation. La fréquence de fonctionnement est donnée par :

$$f = \frac{(V - v_t)^\gamma}{V} \tag{3.4}$$

avec γ une constante, v_t est la tension de seuil.

3.3.1.2 Processeurs à vitesse constante et mode veille :

Les processeurs à vitesse constante opèrent à leur fréquence d'horloge et leur tension d'alimentation nominales et consomment donc la même quantité d'énergie à l'exécution. Le plus souvent, ces processeurs possèdent au minimum deux modes de fonctionnement, le mode actif et le mode veille.
Les algorithmes qui permettent d'utiliser au mieux ce type de processeurs sont appelés DPM (Dynamic Power Management)

3.3.1.3 Processeurs à vitesse variable :

Ce type de processeurs les plus spécifiquement sont conçus pour l'économie d'énergie, ils permettent de varier la tension d'alimentation ainsi que la fréquence de fonctionnement. Les algorithmes de réduction de la consommation d'énergie utilisant ces processeurs sont appelés DVFS (Dynamic Voltage and Frequency Selection).

3.3.2 Méthodes de réduction de la consommation

Afin de pouvoir fonctionner le système embarqué de manière autonome doit disposer d'une source d'énergie, la nature de cette source pouvant influer sur le comportement de système, avec, par exemple, une gestion de la consommation dans le cas des batteries[20].

Pour étendre l'autonomie de fonctionnement d'un système embarqué, il existe deux méthodes : augmenter la quantité d'énergie embarquée ou diminuer la consommation du système.
La première méthode a entrainé de nombreuses recherches dans le domaine des batteries, il est toujours difficile d'augmenter la capacité des batteries sans en augmenter le poids, volume et le cout. La deuxième méthode qui est complémentaire à la première, vise à concevoir un système faible consommation énergétique. Cette méthode utilise des techniques regroupées en trois catégories. Les techniques matérielle, logicielle et les techniques hybride.
La première catégorie est la conception des composants consommant le minimum d'énergie. La deuxième catégorie qui consiste à modifier le logiciel pour diminuer le coût énergétique de son exécution (l'optimisation du code des applications, adaptation des protocoles de communication et d'exécution déporté). Enfin la troisième catégorie, consiste à faire col-

laborer le logiciel et le matériel afin de réduire la consommation totale de l'appareil, et dans ce cas il existe deux façons pour réduire la consommation d'énergie la première est la mise en veille des périphériques, connue sous le terme DPM (pour Dynamic Power Management). La seconde est l'adaptation dynamique de la tension et de la fréquence du processeur, connue sous le terme DVFS (Dynamic Voltage and Frequency Selection). Dans la suite nous présentons les techniques DPM et DVFS.

3.3.2.1 Gestion dynamique de la consommation

La gestion dynamique de la consommation **DPM (Dynamic Power Management)** gérer dynamiquement l'activité du système en faisant des commutations d'un état de veille à un état actif et vice versa. En effet, les techniques DPM permettent de réduire la consommation de l'énergie du système sans dégrader les performances en basculant en mode veille lorsque qu'il n'y a aucune tâche à exécuter et de rebasculer en mode actif quand le processeur est sollicité.
L'objectif de cette technique est de choisir le mode faible consommation du processeur le plus intéressant, lorsqu'il existe des intervalles d'inactivité dans l'ordonnancement ou quand les tâches ne consomment pas la totalité de leur temps d'exécution au pire cas, pour à la fois réduire la consommation et respecter les échéances[21].

3.3.2.2 Adaptation dynamique du voltage et de la fréquence

Variation de tension et de la fréquence du processeur appelée **DVFS (Dynamic Voltage and Frequency Selection)** qui permet de changer dynamiquement la fréquence du processeur ainsi que la tension d'alimentation quand cela est nécessaire. L'objectif de ces méthodes est d'utiliser des processeurs conçus pour réduire l'énergie utilisée en variant la fréquence de fonctionnement et la tension d'alimentation. Dans ce cas l'ordonnancement temps réel consiste à fixer la fréquence de fonctionnement ainsi que la tension d'alimentation.

3.3.3 Ordonnancement temps réel sous contraintes d'énergie :

Plusieurs solutions développées pour réduire la consommation d'énergie dans les systèmes monoprocesseurs adaptant la méthode DVFS (Dynamic Voltage Frequancy Scaling).

Le calcul de vitesse de fonctionnement du processeur s'appuie sur deux approches : approche hors ligne et approche en ligne.

3.3.3.1 Les approches hors ligne

Utilisent les informations disponibles avant l'exécution de l'application. L'objectif est de déterminer la fréquence minimale admissible valable pendent toute la durée de vie du système. Pour les tâches périodiques indépendantes, plusieurs solutions ont été présentées basées sur vitesse par instance des taches, vitesse par tâche et vitesse unique pour toutes les tâches du système. Pour les tâches partageant des ressources critiques. Jejurikar et al ont publiés dans [22] un algorithme de calcul des facteurs de vitesse du processeur. Les auteurs utilisent la condition analytique d'ordonnancement mise en œuvre par Baker dans[23]. Cette condition stipule qu'une configuration de tâches ordonnées en ordre croissant de leurs échéances et qui partagent des ressources critiques soit ordonnançable avec la politique EDF et le protocole PAP (Protocole d'Allocation de la Pile) si la condition suivante est vérifiée :

$$\forall i, i = 1....n, \frac{B_i}{D_i} + \sum_{j=1}^{i} \frac{C_j}{D_j} \leq 1 \tag{3.5}$$

/ B facture de blocage

Où l'algorithme de Jejurikar et al. calcule pour chaque tâche T_i son facteur de vitesse η_i telle que la condition suivante soit vérifiée :

$$\forall i = 1.....n, \frac{B_i}{\eta_i \times D_i} + \sum_{j=1}^{i} \frac{Ci}{\eta_j \times D_j} \leq 1 \tag{3.6}$$

3.3.3.2 Approche en ligne

Vise à déduire la fréquence de fonctionnement du processeur lors de l'exécution réelle des tâches. Cette approche est adéquate dans le cas où les instances d'une tâche ne requièrent pas leur WCET. On distingue deux techniques d'ordonnancement, la première dite "ordonnancement stochastique" qui consiste à trouver pour chaque cycle processeur, la vitesse qui minimise l'espérance de l'énergie en faisant certaines hypothèses probabilistes sur le temps d'exécution. La deuxième technique dite "politiques gain reclaining" qui consiste à utiliser le temps processeur économisé par rapport au WCET pour réduire dans le futur, la vitesse d'exécution d'une ou plusieurs tâches.

3.4 Politiques d'ordonnancement temps réel sous contraintes d'énergie renouvelable :

Il existe deux politiques fondamentales pour ordonnancer des tâches temps réel sous contraintes d'énergie renouvelable, la première basée sur la variation de la vitesse du processeur dite **DVFS** (Dynamic Voltage Frequancy Scaling). La deuxième basée sur la gestion dynamique de la consommation dite **DPM** (Dynamic Power Management).

3.4.1 Algorithmes d'ordonnancement temps réel sous contraintes d'énergie renouvelable :

Dans ce sous section on va présenté les algorithmes classiques et algorithmes avec ordonnancement régulé

3.4.1.1 Algorithme classique

1- FBA (Frame Based Algorithm) : Appelé aussi algorithme d'Allavena et Moussé[24], c'est le premier algorithme proposé pour les systèmes de récupération d'énergie ambiante. Cet algorithme s'applique au modèle de tâches périodiques, signifiant que toutes les tâches ont une période identique et échéance commune. Le principe de FBA consiste à exécuter successivement les tâches d'un même ensemble. Il sélectionne donc les tâches déchargeantes de sorte que le niveau d'énergie disponible dans la batterie baisse plus possible jusqu'à atteindre le niveau minimum E_{min} . Puis il sélectionne les tâches rechargeantes jusqu'à ce que le niveau d'énergie remonte le plus possible jusqu'à atteindre le niveau maximum Emax, maintenant ainsi le niveau d'énergie entre les deux bornes E_min et E_max. Une préemption se produit chaque fois que le niveau d'énergie atteint l'une des deux bornes pour commuter sur le mode de recharge en exécutant des tâches rechargeantes ou de décharge de la batterie en exécutant des tâches déchargeante.

Cet algorithme est optimal pour des tâches périodiques indépendantes. Il suppose la production d'énergie connue initialement. Il est simple à mettre en œuvre et d'avoir une complexité linaire comme les listes des tâches rechargeantes et déchargeantes ne nécessitent pas de tri. Néanmoins, les hypothèses demeurent très restrictives voire impraticables dans des applications réelles. En effet, les auteurs supposent une période et une échéance com-

mune pour toutes les tâches de l'application. De plus, ils considèrent une puissance reçue par la source d'énergie environnementale constante au cours du temps.

2- LSA (Lazy Scheduling Algorithm) : LSA a été introduit en 2006 par Moser et al.[25] de l'institut polytechnique de Zurich. C'est un algorithme piloté par l'énergie (energy-driven algorithm, en anglais). Il permet d'ordonnancer toute configuration de tâches périodiques ou apériodiques ; ces tâches sont exécutées par un monoprocesseur alimenté par un réservoir d'énergie rechargeable par une source d'énergie renouvelable, ce réservoir d'énergie reçoit une puissance instantanée pouvant variée au cours du temps.
A chaque réveil d'une tâche, l'ordonnanceur calcule une date de démarrage propre à cette tâche. Cette date représente l'instant à partir duquel la tâche peut commencer à s'exécuter sur le processeur en utilisant la puissance de consommation maximum Pmax pendant toute son exécution. La détermination de cette date faite de telle sorte que le processeur ne commence à l'exécuter ni trop tôt, ni trop tard. Entre la date d'arrivée et la date de démarrage, le processeur est laissé volontairement en veille pour permettre au réservoir de se charger. Cette intervalle de temps de recharge est calculé tel que, lorsque la tâche commence à s'exécuter, le processeur disposera du suffisamment d'énergie pour fonctionner de façon continue jusqu'à la date d'échéance de la tâche. LSA est un algorithme clairvoyant du point de vue énergétique ; donc il suppose avoir une connaissance du profil énergétique de la source. LSA est un algorithme conduit par priorité dynamique piloté par l'énergie récupérée de l'environnement. Quand nous considérons une architecture monoprocesseur en charge d'exécuter des tâches soumises à des échéances strictes et qui requièrent des quantités d'énergie différentes pour leur exécution, il peut garantir perpétuellement leurs contraintes temporelles en exploitant de façon adéquate à la fois la ressource processeur et la ressource énergie ambiante. Même si LSA est optimal, il présente quelques inconvénients. En effet, les hypothèses faites sur la configuration de tâches sont très restrictives. L'énergie consommée par une tâche est supposée proportionnelle à sa durée d'exécution. Or, l'énergie totale que consomme une tâche au cours de son exécution n'est pas proportionnelle à la durée de son exécution. Cette énergie dépend en particulier des différents circuits électroniques dont aura besoin la tâche.

3- EDeg (Earliest Deadline with energy guarantee) : EDeg est proposé dans (Chetto et al.,2011, El Ghor et al.,2011)[26, 27]. Cet algorithme vise à ordonnancer les

tâches selon EDF ; gère les intervalles de passivité du processeur selon la disponibilité d'énergie. Avant d'autoriser une instance à s'exécuter, EDeg utilise la notion de la laxité énergétique pour prédire d'éventuels futurs dépassements d'échéances dus à une insuffisance d'énergie. La laxité énergétique représente la quantité maximale d'énergie qui peut être consommée sans tomber dans une situation d'épuisement énergétique. Les auteurs considèrent un système composé d'un monoprocesseur alimenté par un réservoir d'énergie rechargé par une source d'énergie renouvelable. Le processeur doit être capable d'exécuter une configuration de tâches périodiques selon EDF tout en considérant leurs besoins énergétiques, leurs échéances et l'énergie disponible dans le réservoir.
Ainsi à chaque instant, une tâche n'est s'exécutée que si le réservoir n'est pas vide et qu'il y a d'énergie suffisante dans le système ou si le réservoir est vide, le processeur doit être mis en veille pour permettre de recharger la batterie.

4- EA-DVFS : Algorithme de sélection dynamique de tension et de fréquence (EA-DVFS) sensible à l'énergie est proposé dans (Liu et al., 2008)[28]. L'algorithme EA-DVFS ajuste le comportement du processeur en fonction de la somme de l'énergie stockée et de l'énergie récoltée dans une durée ultérieure. Spécifiquement, si le système a suffisamment d'énergie, les tâches sont exécutées à pleine vitesse ; sinon, le processeur ralentit l'exécution des tâches pour économiser de l'énergie. Les résultats de la simulation montrent que lorsque l'utilisation est faible, l'algorithme EA-DVFS donne un taux d'échéance au moins 50% inférieur à celui de la politique de planification différée. De même, lorsque la charge de travail est faible, la taille de stockage minimale est réduite d'au moins 25%.

5- HA-DVFS : Algorithme de gestion de la consommation prenant en compte la récolte qui vise à obtenir une efficacité énergétique et des performances système optimales dans les systèmes temps réel de récupération d'énergie. L'algorithme proposé dans (Liu et al)[29] utilise des techniques de planification statiques et adaptatives combinées à une sélection dynamique de tension et de fréquence pour obtenir de bonnes performances du système en cas de contraintes de temps et d'énergie. Il permet d'améliorer les performances du système en exploitant le mou de la tâche avec une sélection dynamique de la tension et de la fréquence et en minimisant le gaspillage de l'énergie récupérée. Cet algorithme améliore la performance du système en cas d'échec de délai et la capacité de stockage minimale requise pour un taux de non-respect des délais. En comparaison avec les algorithmes

existants, l'algorithme HA-DVFS atteint de meilleures performances en termes de taux de non-respect des délais et de capacité de stockage minimale dans divers paramètres de charges de travail et de profils d'énergie collectés.

3.4.1.2 Algorithmes d'ordonnancement temps réel régulé sous contraintes d'énergie renouvelable :

1- AS-DVFS (Adaptive Scheduling and DVFS algorithm) proposé dans (Shaobo et al.,2009)[30], dans lequel le réglage de la fréquence de fonctionnement du processeur tient compte des contraintes temporelles et des contraintes d'énergie. L'efficacité énergétique est atteinte en exploitant les laxités des tâches pour récupérer de l'énergie en initialisant la séquence d'ordonnancement selon LSA puis en distribuant uniformément la charge de travail au fil du temps.

D'autre algorithmes ont été proposé dans la literature comme :

2- L'algorithme EDfbs-eg (Earliest Deadline Feedback Scheduling with energy guarantee) est proposé dans (Abbas et al., 2013)[31]. Il s'agit de l'application du principe de l'ordonnancement temps réel régulé au contexte des systèmes récupérateurs d'énergie environnante.

3- l'algorithme FS-EH (Feedback Scheduler for Energy Harvesting) proposé dans (Abbas et al., 2014, Abbas et al., 2016)[32, 33], il apporte des solutions aux points faibles de l'algorithme EDfbs-eg, et cet algorithme prend en compte la quantité d'énergie disponible dans la batterie, la variabilité des durées d'exécution des tâches et les besoins courants du processeur.

4- L'algorithme FSCE-EH proposé dans (Abbas et al., 2015)[34] permis d'exécuter des tâches avec une vitesse juste supérieure à la vitesse actuelle si le coût de contrôle maximal du système est supérieure à 0. Il renvoie une vitesse juste en dessous de la vitesse actuelle lorsque le coût de contrôle maximal du système est nul et que le niveau d'énergie disponible est inférieur à la valeur de seuil.

3.5 Conclusion :

Plusieurs travaux ont été proposé ces dernières années pour réduire la consommation d'énergie à travers le réglage de la vitesse du processeur sous l'utilisation de la commande de rétroaction ou l'ordonnancement temps réel régulé, et d'autres travaux de recherche qui prenne en compte l'énergie renouvelable ou ambiante. La prise en compte des ressources critiques reste une problématique qui sera traitée dans le chapitre 4, où les solutions proposées considèrent les incertitudes des durées d'exécution des tâches et l'utilisation de l'énergie ambiante et sous contrainte des ressources critique.

Deuxième partie

Contribution

Chapitre 4

L'ordonnancement temps réel régulé sous contrainte des ressources critique et l'énergie renouvelable

4.1 Introduction

Plusieurs travaux de recherche traitent le problème de l'ordonnancement temps réel des tâches sous contrainte d'énergie ainsi que l'ordonnancement temps réel régulé dans les systèmes embarqués ont été largement étudié dans la littérature. Toutefois, les travaux réalisés ne prennent pas en compte l'ordonnancement temps réel régulé dans le contexte des tâches périodiques sous contraintes de ressources, d'énergie renouvelable. Ainsi, nous traitons dans cette contribution le problème du contexte sus cité.

L'objectif d'un système autonome est d'assurer un fonctionnement perpétuel sans l'intervention humaine et ceci grâce à des batteries (ou tout autre type de réservoirs d'énergie), qui se rechargent en continu au cours de temps à partir d'une source d'énergie renouvelable. Pour cela, la réduction de la consommation d'énergie de ces systèmes embarqués est des métriques d'optimisations cruciales pendant la conception et la réalisation de tels systèmes embarqués. Ces systèmes nécessitent souvent des tâches temps réel périodiques, où certaines sont à échéances critiques.

4.2 Hypothèses :

nous proposons dans cette étude de la modélisation les hypothèses suivantes :

— L'ensemble des tâches à ordonnancer ainsi leurs caractéristiques sont connus ;

— Les tâches à ordonnancer sont périodiques et partagent des ressources critiques accessibles en exclusion mutuelle.

— L'ensemble des tâches est ordonnancé suivant la politique d'ordonnancement EDF.

— Nous estimons que le système est alimenté par un système de récupération de l'énergie ambiante.

4.3 Modélisation de système :

Nous considérons un système de contrôle (voir la figure 4.1) composé de plusieurs tâches partageant un processeur avec une tension et une fréquence variables. Chaque tâche est responsable du contrôle d'un procédé. On suppose que toutes les tâches sont périodiques et indépendantes les unes des autres.

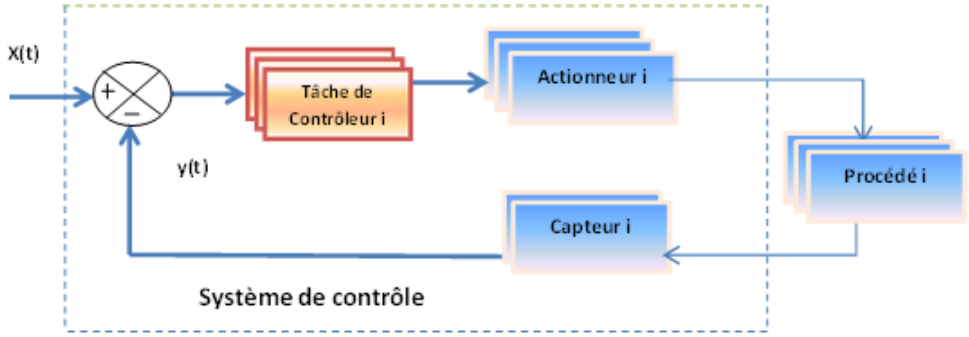

FIGURE 4.1 – Schéma d'un système embarquée temps réel de contrôle

Une tâche périodique t_i est modélisée par les quatre paramètres chronologiques de base suivants : (r_i, C_i, D_i, P_i) avec :

— r_i, le moment de la première requête d'exécution de la tâche (temps initial)

— $C_i = C_{max}$, la durée d'exécution maximale de la tâche, quand elle dispose du processeur pour elle (le coût d'exécution), Cette quantité est appelé généralement WCET (Worst Case Exécution Time) ; On précisons ici que les tâches peuvent s'exécutées

avec des durées inférieurs à leurs WCET ($C_i \leq C_i$). Dans le but d'avoir ces durées, nous utiliserons par la suite un générateur aléatoire de durées d'exécution.

— D_i le délai critique acceptable pour l'exécution de la tâche (l'échéance), et l'échéance $d_{i,k}$ de la k_{eme} instance est définie par :$d_{i,k} = r_{i,k} + D_i$

— P_i la période d'exécution, écart entre deux activations successives d'une tâche périodique, et la date d'activation r_k du la k_{eme} instance est définie par :
$r_{i,k} = r_0 + K \times P$

Le facteur d'utilisation du processeur par l'ensemble des tâches périodiques sous la politique EDF est :

$$U = \sum_{i}^{n} \frac{C_i}{P_i} \tag{4.1}$$

La tâche FBS c'est la tâche la plus prioritaire, peut préempte tous les autres tâches, et elle est responsable de calculer et de changer la vitesse du processeur. Donc toute instance $t_{i,k}$ préemptée par la tâche FBS peut être exécutée avec deux ou plusieurs vitesses du processeur.

Nous assumons que la tâche FBS implémentant le régulateur d'ordonnancement s'exécute périodiquement. Pour la partie caractérisant le régulateur d'ordonnancement (FBS), nous utilisons les notations suivantes Cervin [17] :

— $\hat{C}_i$, est la durée d'exécution estimée de la tâche t_i sous la vitesse maximale du processeur ;

— $\hat{C}_{i,S}$, est la durée d'exécution estimée actuelle de la tâche t_i lorsque la vitesse du CPU est réduite par S, c-à-d : $\hat{C}_{i,S} = \hat{C}_i/\mathrm{S}$;

— c_i la durée d'exécution actuelle de l'instance $t_{i,k}$ sous l'actuelle vitesse du processeur S ;

Dans la suite, nous utilisons le terme "actuelle" pour désigner une quantité mesurée (le temps d'exécution à la fin d'une instance), et le terme "estimée" est utilisé pour désigner une valeur prédite basée sur une fonction historique (par exemple, l'équation ...).

4.4 Modélisation de partage des ressources critiques

On suppose que le système possède un ensemble de ressources critiques partagées par les tâches périodiques. les tâches utilisent des éléments mis en commun au niveau du

système. Certains ces ressources comme les zones mémoires, ne sont pas ou ne doivent pas être accessibles, par plus d'une tâche à la fois, elles sont dites ressources critiques. Une tâche t_i de durée totale C_i qui utilise une ressource critique R_k avec des paramètres ($\alpha_{i,k}$, $\beta_{i,k}$,$\gamma_{i,k}$) possède dans son code une zone protégée, appelée section critique, pendant laquelle elle accède à cette ressource. Cette section critique est protégée par des primitives permettant de gérer l'exclusion mutuelle comme un sémaphore. Dans le cas de l'ordonnancement temps réel régulé, où la durée d'exécution est variable, une tâche t_i de durée d'exécution estimée $\hat{C}_i$ et qui utilise des ressources critiques R1, R2,...,Rn, est modélisée sous la forme :

$$< r_i, \hat{C}_i, D_i, P_i, (\hat{\alpha}_{i,1}, \hat{\beta}_{i,1}, \hat{\gamma}_{i,1}), ..., (\hat{\alpha}_{i,n}, \hat{\beta}_{i,n}, \hat{\gamma}_{i,n}) > \tag{4.2}$$

où chaque ressource R_k est définie par le triplet ($\alpha_{h,k}$, $\beta_{h,k}$,$\gamma_{h,k}$) :

— $\alpha_{h,k}$ temps avant la section critique estimé ;

— $\beta_{h,k}$ durée de la section critique estimée (ressource utilisée) ;

— $\gamma_{h,k}$ temps après la section critique estimé.

Ces trois valeurs doivent satisfaire à l'égalité suivante :

$$\hat{C}_h = (\alpha_{h,k} + \beta_{h,k} + \gamma_{h,k}) \tag{4.3}$$

Où $\hat{C}_h$ représente la durée d'exécution estimée de la tâche périodique qui accède à la ressource critique R_k. Si une tâche utilise plusieurs ressources critiques, alors ces sections doivent être correctement imbriquées.

On suppose que les tâches partagent une ou plusieurs ressources critique, elles peuvent passer dans l'état bloqué afin de permettre la garantie d'exclusion mutuelle. Si une tâche t_j utilise une ressource critique Rk et si une autre tâche t_i plus prioritaire demande l'utilisation de la même ressource, la tâche t_i sera bloquée jusqu'à ce que la ressource soit libérée. Ceci implique que la mise en concurrence de plusieurs tâches pour l'accès à une ressource engendre des retards d'exécution qu'on doit intégrer dans les durées d'exécution des tâches lors de l'analyse de l'ordonnancement. On appelle la durée pendant laquelle la tâche t_i reste bloquée le temps de blocage B_i. Donc le temps de blocage maximum d'une tâche peut être calculé.

4.5 Prise en compte des contraintes de ressources

Pour la prise en compte des contraintes de ressources, on a opté à l'utilisation de protocole d'allocation de la pile SRP (Stack Resource Protocol) proposé par Baker dans [23]. Ce protocole permet de gérer des ressources à plusieurs instances. Il est utilisé dans le contexte de tâches à échéances contraintes (où $D_i \leq P_i$). Les principes de base de ce protocole sont :

1. En plus de sa priorité, chaque tâche se voit attribuer, de façon statique (hors ligne), un paramètre π appelé niveau de préemption. Ce dernier permet de prédire les éventuels blocages. Les niveaux de préemption assignés inversement proportionnelle aux échéances des tâches ou bien $\pi_i = \frac{1}{D_i}$.

2. Chaque ressource se voit assigner dynamiquement une valeur plafond notée CR_k, déterminée par la valeur maximale des niveaux de préemption des tâches actives ayant besoin de plus d'instances de la ressource R_k, c-à-d, les tâches dont la demande en unités de sémaphore ne peut pas être satisfaite.

$$CR_k = max_i\{\pi_i | t_i \, se\, bloque\, en\, (ou\, besoin\, de)\, R_k\} \tag{4.4}$$

3. Le plafond dynamique du système noté Π, est la valeur maximale des plafonds des ressources en cours d'utilisation qui est défini comme :

$$\Pi(t) = max[\{CR_k | R_k\, est\, actuellement\, occupé\} \cup \{0\}] \tag{4.5}$$

Une tâche t_i, peut préempter une autre tâche t_j si les conditions suivantes sont vérifiées :

- L'échéance absolue D_i de t_i est inférieure à celle de t_j ;
- Le niveau de préemption de t_i est supérieur à celui de t_j : $\pi_i > \pi_j$
- Le niveau de préemption de t_i est supérieur au plafond système : $\pi_i > \Pi$.

Pour analyser l'ordonnancement d'une configuration de n tâches périodiques sous contraintes de ressources, nous nous sommes basé sur la condition suffisante d'ordonnancement établi par Baker dans [23], que nous adaptons suivant la formule :

$$\forall i = 1, \ldots, n \sum_{j=1}^{i} \frac{\hat{C}_j}{D_j} + \frac{B_i}{D_i} \leq 1 \tag{4.6}$$

Où B_i représente le temps de blocage maximum estimé de la tâche t_i. Cette durée égale à la plus longue section critique des tâches périodiques, dont le niveau de préemption est inférieur à celui de t_i, et qui utilisent une ressource ayant la valeur plafond supérieure ou égale au niveau de préemption de t_i. Ainsi, le pire temps de blocage d'une tâche t_i s'exprime par :

$$B_i = max\{\hat{\beta}_{i,k} | \pi_i > \pi_j \wedge CR_k \geq \pi_i\} \tag{4.7}$$

Où $\hat{\beta}_i$ est le temps de la section critique estimé.

$$\hat{\beta}_i(k) = \lambda\hat{\beta}_i(k-1) + (1-\lambda)\beta_i \tag{4.8}$$

β_i temps de la section critique réel.

4.5.1 Modèles de la consommation d'énergie

Nous intéressant à la stratégie DVFS qui permet de changer dynamiquement la fréquence du processeur ainsi que la tension d'alimentation quand cela est nécessaire, Cette stratégie est motivée par l'apparition de circuits électroniques à tension d'alimentation et fréquence variables. L'idée est d'utiliser les informations concernant les caractéristiques des tâches et la politique d'ordonnancement pour dériver la vitesse notée S du processeur (le rapport entre la fréquence actuelle f et la fréquence maximale du processeur f_{max}, c-à-d. S = f/f_{max}) qui minimise la consommation totale d'énergie en conformité avec toutes les échéances et les autres contraintes des tâches. Par exemple si le facteur d'utilisation du processeur U est inférieur à 1 et que ce même processeur exécute les tâches avec une vitesse maximale S_{max}, il est possible de réduire à la vitesse du processeur S (ou à la fréquence f) (et donc d'augmenter les temps d'exécution des tâches) jusqu'à une vitesse qui donne un facteur d'utilisation égal à 1. Nous donnons dans ce qui suit les deux modèles de consommation d'énergie que nous avons utilisés dans nos contributions : le modèle théorique et le modèle empirique.

4.5.1.1 Modèle théorique

Le circuit CMOS (Complementary Metal Oxide Semiconductor) est la technologie dominante dans les circuits électroniques[19] L'énergie consommée par un processeur dans un intervalle de temps [a, b] est par définition l'intégrale de la puissance dissipée :

$$E = \int_a^b P(t)\,\mathrm{d}t \tag{4.9}$$

où P(t) est la puissance dissipée à l'instant t. Cette puissance est décomposée en deux types, la puissance statique et dynamique

$$P = P_{dyn} + P_{sta} \tag{4.10}$$

La puissance statique P_{sta} est due à des courants de fuite des transistors et la puissance dynamique P_{dyn} est due à l'activité de commutation :

Dans le circuit CMOS la puissance statique du processeur n'a pas été prise en compte car cette dernière reste négligeable, et donc la puissance s'exprime par :

$$P \approx P_{dyn} \approx \alpha \times c \times f \times V^2 \tag{4.11}$$

Où α est le nombre de transitions par cycle d'horloge, c est la somme des capacités dans le circuit chargées et déchargées à chaque cycle, f la fréquence nominale du processeur et V la tension d'alimentation. La fréquence de fonctionnement est donnée par :

$$f = \frac{(V - v_t)^\gamma}{V} \tag{4.12}$$

avec γ une constante, v_t est la tension de seuil.

4.5.1.2 Modèle empirique

Ce modèle se base sur l'observation du système pour déduire d'autres modèles. Ainsi, notre étude est basée sur le processeur commerciale XScale. Ces principales caractéristiques sont données dans le tableau ci-dessous 4.1 [35].

V(V)	0,75	1,0	1,3	1,6	1,8
$f(MH_z$	150	400	600	800	1000
S_m	0,15	0,4	0,6	0,8	1
$P(mW)$	80	170	400	900	1600
Énergie de commutation (1.2 μ J)					
Temps de commutation (12 μ s)					
Puissance en mode veille (mW) (63.85))					

TABLE 4.1 – Les paramètres du processeur XScale

consommation continue qui soit proche du modèle discret. Selon la relation dérivée dans [35], plusieurs modèles de puissance du processeur XScale sont cités dans la littérature.

Ces modèles décrivent la puissance consommée sous forme d'une fonction polynomiale de la vitesse du processeur S. Nous avons choisi d'utiliser le modèle présenté dans [36] où la puissance est écrite comme :

$$P = 1543,28 \times S^{2,87} + 63,85 \tag{4.13}$$

La figure 4.2 montre que l'approximation continue de la puissance consommée, en fonction des vitesses, s'accorde avec puissance discrète du constructeur. Par conséquent, la fonction montrée dans la figure 4.2 semble être plus réaliste, si nous supposons l'existence d'un processeur XScale capable de changer sa fréquence en continu.
Ainsi, dans un intervalle du temps [t1, t2], l'énergie consommée est donnée par : $Ec(t1, t2) = \int_{t_1}^{t_2} P(S(t))dt$.

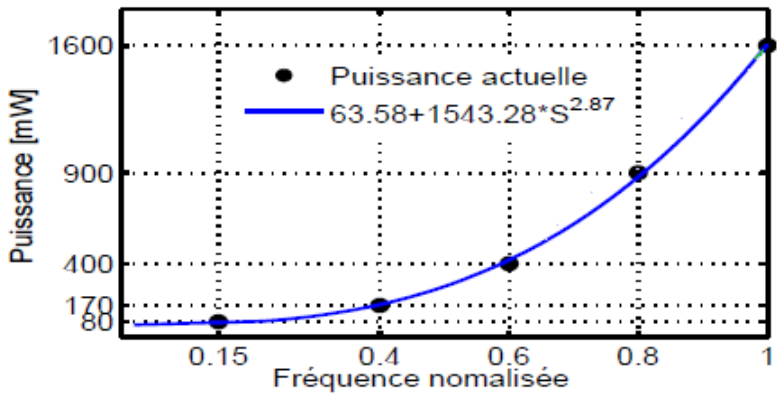

FIGURE 4.2 – La fonction de la puissance consommée

4.5.2 Modélisation de la source d'énergie :

On suppose que l'énergie de l'environnement, telle que l'énergie solaire, est récupérée et transformée en énergie électrique alimentant la batterie d'un système embarqué. Le comportement d'une source d'énergie solaire est modélisé comme suit [37]

$$P_s = |0,9 \times R(t) \times \cos(\frac{t}{0,7\pi}) \times \cos(\frac{t}{0,1\pi})| \tag{4.14}$$

Où R(t) est une variable aléatoire uniforme distribué entre 0 et 1. Les valeurs de P_s sont limitée à une valeur maximale égale à P_s,max = 0,9.voir figure 4.3)

Comme montrer sur la figure 4.3, la courbe de puissance obtenue P_s(t) simule des périodes similaires à celles obtenues avec des cellules solaires dans un environnement

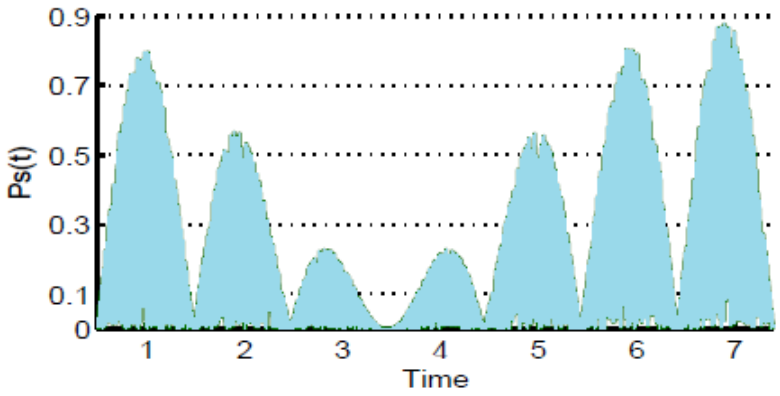

FIGURE 4.3 – La courbe de la puissance Ps(t)

extérieur. La puissance récupérée P_s(t) est la puissance nette alimentant batterie. L'énergie totale, Es(t1, t2), récupérée dans un intervalle du temps [t1, t2] est donnée par :

$$E_s = \int_{t_1}^{t_2} Ps(t)\,\mathrm{d}t \tag{4.15}$$

4.5.3 Modélisation du réservoir d'énergie :

Le système considéré utilise une unité de stockage de l'énergie d'une capacité nominale (E, exprimé en Joules ou Watt-heure). Le niveau d'énergie, noté E(t) à un moment donné t, doit rester entre deux limites E_{min} et E_{max} donnant une capacité de l'énergie disponible pour le processeur égale à E = E_{max} − E_{min}. Si l'unité de stockage de l'énergie est complètement déchargée c-à-d le niveau d'énergie égal à E_{min}, dans ce cas l'énergie est épuisée, aucune tâche ne peut être exécutée et le processeur doit s'arrêter et restera inactif. Si l'élément de stockage de l'énergie est complètement plein c.-à-d. son niveau d'énergie égal à E_{max}, et nous continuons à le charger, l'énergie est gaspillée. Pour réduire le gaspillage et assurer au même temps une qualité de contrôle des tâches, il serait utile d'exécuter les tâches avec la vitesse maximale du processeur lorsque la batterie est presque pleine.

4.6 Ordonnancement temps réel régulé sous contrainte des ressources et des systèmes de récupération des énergies

Dans cette section nous considérons le problème d'ordonnancement des tâches périodique synchrones au contexte des systèmes de récupération d'énergie. L'objectif est d'obtenir l'ordre d'exécution des tâches qui garantit le respect de synchronisation à moins consommation d'énergie.

Nous considérer un système entièrement autonome capable de fonctionner dans le respect de ses contraintes temporelle et énergétique. Le système considéré est illustré par la figure 4.4. C'est un système temps réel de récupération de l'énergie composé d'un monoprocesseur alimenté par une batterie qui est rechargée par une énergie renouvelable, ce processeur exécute simultanément un ensemble N de tâches de contrôle qui sont indépendantes et préemptables. Chaque tâche est responsable de contrôler un procédé physique indépendant. En plus des boucles de contrôle, une boucle de réaction extérieure est ajoutée au système pour implémenter la rétroaction. Le rôle principale d'un ordonnanceur par rétroaction (boucle fermée) est l'utilisation des informations mesurées sur le système informatique pendant son exécution, comme le niveau de l'énergie disponible et l'utilisation du processeur pour améliorer l'ordonnancement des tâches par le respect de leurs échéances et pour optimiser la consommation énergétique par le calcule de bon facteur de vitesse du processeur.

4.6.1 Calcule le facture de vitesse

Nous présentons dans cette sous section un algorithme (voir algorithme 2) qui permet de calculer le facteur de vitesse du processeur noté Sf et applique le principe de l'ordonnancement temps réel régulé au contexte des ressource et des systèmes récupérateurs d'énergie renouvelable.

Le facteur Sf est calculé de telle sorte que la condition analytique suivante soit vérifiée :

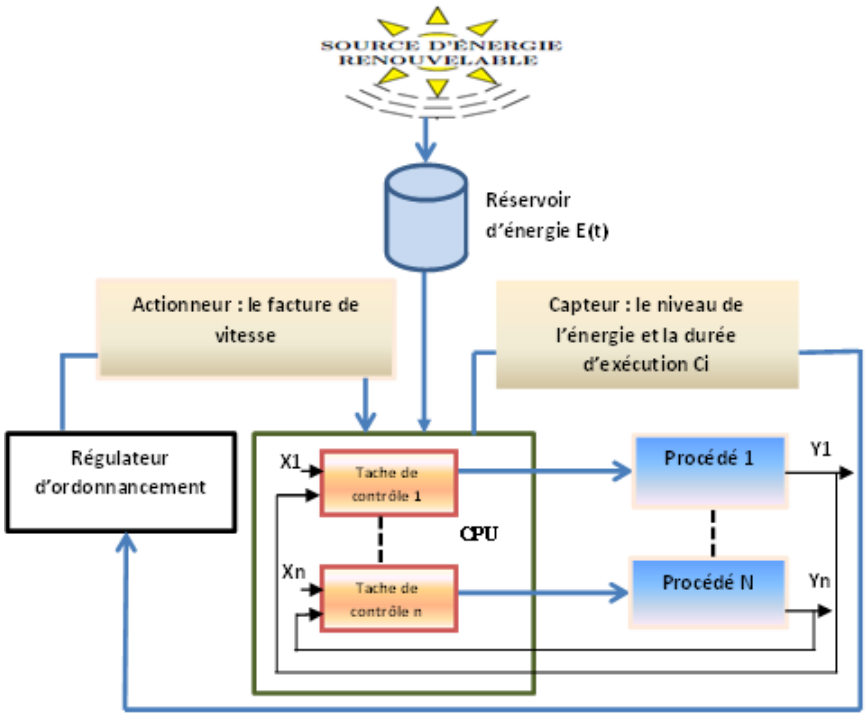

FIGURE 4.4 – Régulateur d'ordonnancement

$$\forall i = 1....n; \sum_{j=1}^{i} \frac{\hat{C}_j}{D_j} + \frac{B_i}{D_i} \leq 1 \tag{4.16}$$

La solution proposée nous permet d'exécuter les tâches avec la vitesse du processeur maximale si le système est surchargé ou la quantité d'énergie disponible dans la batterie est supérieure à un seuil (fixé hors ligne). Cela permet de réduire le risque de dépassements d'échéances. Si l'énergie disponible est inférieure à ce seuil, la vitesse du processeur discrète retournée est proportionnelle à l'énergie disponible et à l'utilisation instantanée du processeur. Ceci favorise la recharge de la batterie et évité le risque de surcharger, puisque le taux d'énergie disponible dans la batterie est souvent supérieur à l'utilisation instantanée du processeur.

4.6.1.1 Hypothèses

nous supposons que un ensemble des n tâches périodiques ordonnées par ordre croissant de leurs échéances avec leurs caractéristiques ($\hat{C}_i$, D_i, P_i, B_i). Et nous supposons que le processeur fonctionne avec Q fréquences discrète, où f_q : $\{f_q \mid 1 \leq q \leq Q$, $f_{min} = f_1 < ... < f_q = f_{max}$. Nous définissions le facteur de vitesse du processeur Sf et comme

la fréquence normalisée f_q avec le respect de la fréquence maximale f_{max}, ceci est donné par :

$$Sf = f_q / f_{max} \tag{4.17}$$

4.6.1.2 Estimation des durées d'exécution

Dans notre cas, utilisation de l'ordonnancement régulé (Feedback Scheduler) selon de Cervin [17], la durée d'exécution est elle-même la mesure utilisée dans la boucle. Un filtre passe bas avec un facteur d'oubli λ dans intervalle (0,1) est utilisé pour estimer la durée d'exécution sur laquelle on doit se baser pour calculer le facteur d'utilisation du processeur. L'estimation de la durée d'exécution de chaque tâche sur laquelle le régulateur d'ordonnancement peut agir est comme suit : Nous initialisons l'estimateur de chaque tâche à sa pire durée d'exécution des tâches (WCET). L'estimateur associé à la tâche t_i est donné comme suit :

$$\begin{aligned} \hat{C}_i(0) &= WCET_i \\ \hat{C}i(k) &= \lambda \hat{C}_i(k-1) + (1-\lambda)c_i \end{aligned} \tag{4.18}$$

Où les c_i sont les durées d'exécution réelles des tâches et $\hat{C}_i(0)$ est l'initialisation arbitraire de l'estimateur de chaque tâche pire à sa période nominal. λ permet de lisser l'estimateur dans le temps, s'il est élevé (λ=1) l'estimateur prendra plus de temps pour s'adapter à un changement soutenu du temps d'exécution de la tâche, par contre s'il est faible (proche de 0), il sera très sensible à des changements transitoires.
Ainsi, lorsque le $k^{ème}$ instance de la tâche t_i termine l'exécution, elle renvoie sa durée d'exécution c_i qui sera utilisée pour estimer la prochaine durée d'exécution $\hat{C}_{i,K}$ puis le facteur d'utilisation $\hat{U}$ comme suit :

$$\hat{U} = \sum_{i=1}^{n} \frac{\hat{C}_i(k)}{D_i} \tag{4.19}$$

Pseudo-code de l'algorithme d'estimation des durées d'exécution : Étant donné C_i la durée d'exécution actuelle de l'instance $t_{i,k}$, et $WCET_i$ pire durée d'exécution de tâche t_i. La durée d'exécution estimée est donnée par l'algorithme ci-dessous :

Algorithm 1: Estimation des durées d'exécution $\hat{C}_i(k)$

Input:

c_i, la durée d'exécution actuelle de l'instance $t_{i,k}$;

$\hat{C}_i(k-1)$, la durée d'exécution estimé de l'instance (k-1);

Result:

$\hat{C}_i(k)$, la durée d'exécution estimé de l'instance k;

1 $\hat{C}_i(k) \leftarrow \lambda \hat{C}_i(k-1) + (1-\lambda)c_i$;

2 **return** $\hat{C}_i(k)$

Considérons le système avec un régulateur d'ordonnancement et considérons aussi l'utilisation instantanée du processeur comme suit :

$$U_{inst}(t) = \sum_{i}^{n} \frac{\hat{C}_i}{D_i} \tag{4.20}$$

Nous adaptons le test hors ligne de la politique d'ordonnancement classique EDF à l'invocation de la tâche régulateur d'ordonnancement (tâche FBS) à l'instant t comme $U_{inst}(t) \leq 1$

4.6.1.3 Principes de l'algorithme

L'algorithme permet d'exécuter les tâches avec la vitesse du processeur supérieurs à la vitesses actuelle si le système est surchargé (dépassement d'échéance $U_{inst}(t) > 1$) sinon lorsque la quantité d'énergie disponible $E(t)$ dans la batterie est supérieure à un seuil L ($E(t) > L$) (fixé hors ligne) le processeur continuera le fonctionnement avec la vitesse actuelle. Si l'énergie disponible est inférieure à ce seuil et que le système n'est pas surchargé, la vitesse du processeur discrète retournée est proportionnelle à l'énergie disponible et à l'utilisation instantanée du processeur.

Pseudo-code de l'algorithme Étant donné C_i la durée d'exécution actuelle de l'instance $t_{i,k}$, et $WCET_i$ pire durée d'exécution de tâche t_i. Le facteur de vitesse est calculé par l'algorithme FBS-resources (voir l'algorithme 2)

Algorithm 2: FBS-resources

Input:

$\{\tau_1, ..., \tau_1\}$ l'ensemble des n tâches périodiques ordonnées par ordre croissant de leurs échéances avec leurs caractéristiques ($C_i,\ D_i,\ T_i,\ B_i$);

$El(t)$, la quantité d'énergie disponible à l'instant t;

$SF_{acct}(k)$, le facteur de vitesse acttuel;

L, le seuil de la batterie;

Result:

SF, le facteur de vitesse

$X \leftarrow 0, Y \leftarrow 0, i \leftarrow 1, V \leftarrow 0$ {initialisation};
while $i \leq n$ **do**
  $j \leftarrow 1$;
  while $j \leq i$ **do**
    $X+ = C_j/D_j$;
    $j \leftarrow j + 1$;
  end
  $Y \leftarrow B_i/D_i$;
  $U \leftarrow X + Y$;
  if $U > 1$ **then**
    $V \leftarrow max(V, SF_{acct}(k + 1))$
  end
  else
    if $El(t) > L$ **then**
      $V \leftarrow max(V, SF_{acct}(k))$
    end
    else
      $V \leftarrow min\{SF_m | SF_m \geq max(V, U, El(t)/L)\}$;
    end
  end
  $X \leftarrow 0$;
  $i \leftarrow i + 1$;
end
$SF \leftarrow V$;
return SF ;

4.7 Évaluations et Résultats :

Dans cette section, nous présentons les simulations qui ont été menées afin d'évaluer les propositions décrites précédemment. Nous y présentons aussi les résultats expérimentaux réalisés. Nous commençons par présenter l'environnement de simulation. Puis, nous présentons les résultats obtenus ainsi que l'analyse qui a été faite.

4.7.1 Environnement de simulation :

Il existe plusieurs simulateurs pour les systèmes de contrôle temps réel, en particulier TrueTime qui est compatible avec Matlab /Simulink. Le choix s'est porté sur la boite à outils TrueTime car celle-ci en open source et utilisé par la plus part des chercheurs dans le domaine des systèmes temps réel.

TrueTime : est un simulateur basé sur Mtlab/Simulink pour les systèmes de contrôle en temps réel. Développé depuis 1999 par des chercheurs Suèdois de Lund. Ce simulateur fonctionne sous la forme d'une bibliothèque [38]utilisable sous Simulink comme le montre la figure 4.5.

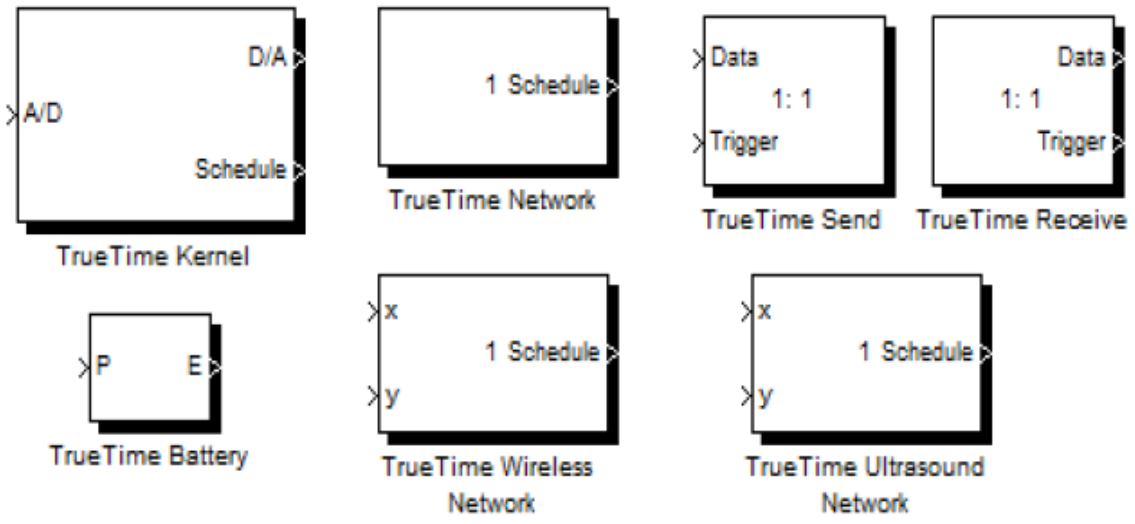

FIGURE 4.5 – Bibliothèque de TrueTime 2.0

Cet outil est composé de plusieurs blocs :

— Un noyau temps réel ;

— Réseau True Time ;

— Nœud d'émetteur et un nœud récepteur ;

— Réseau sans fil ;

— Réseau d'ultrason ;

— Une batterie.

True Time Fournit une collection de fonctions Matlab utilisées pour, par exemple, modéliser les tâches de contrôle et modifier des attributs de tâches. Cet outil permet aussi d'analyser l'ordonnançabilité et affiche les états d'ordonnancement (runing, preempted, sleeping) des tâches simulées. True Time facilite la modélisation de régulateurs embarqués et distribués.

Pour effectuer les simulations nous avons utilisé l'outil TrueTime, dans lequel nous avons implémenté le protocole de gestion des ressources critiques SRP et le algorithme FBS-resource que nous avons développé.

Nous considérons un système de contrôle embarqué composé de trois boucles de contrôle indépendantes (voir la figure 4.6). Chaque procédé est contrôlé en utilisant l'algorithme PID dont les paramètres sont ceux conçus et proposés dans [38]. La fonction de transfert de chaque procédé est : G(s) = 1000/(s2+s). Le signal de référence est un signal carré avec une fréquence 1 Hz et une amplitude de 1 unité. Nous avons implémenté un générateur aléatoire des durées d'exécution des tâches c_i selon la loi de Weibull donnée ci-après : c_i = - $\log(1-R)^{\frac{1}{K}} \times \beta$
avec k = 3, β= 0.0015, et R une variable aléatoire uniformément distribué entre 0 et 1. Nous choisissons ces paramètres afin que le maximum des c_i soit inférieur ou égal à $WCET_i$.

Le but de ce générateur est de simuler une distribution typique des durées d'exécution actuelles c_i qui soit proche de la répartition donnée.
Nous considérons un ensemble de trois tâches de contrôle T =$\{t_i | 1 \leq i \leq 3\}$ et t_i = ($WCET_i$,D_i,P_i) avec $WCET_i = 3$ ms pour chaque tâche t_i où les périodes d'échantillonnage des trois boucles : t1 = 15 ms, t2 = 16 ms, et t3 = 17 ms, respectivement. Nous considérons que la puissance consommée par une tâche t_i sous la vitesse S est donnée par l'équation : $P = 1543,28 \times S^{2,87} + 63,85$. Nous supposons que la charge de la batterie à l'instant t =

0 est E_{max} = 2,5 J, et la période de la tâche FBS est égale à 80 ms et nous choisissons le facteur d'oubli de comme $\lambda = 0{,}9$.

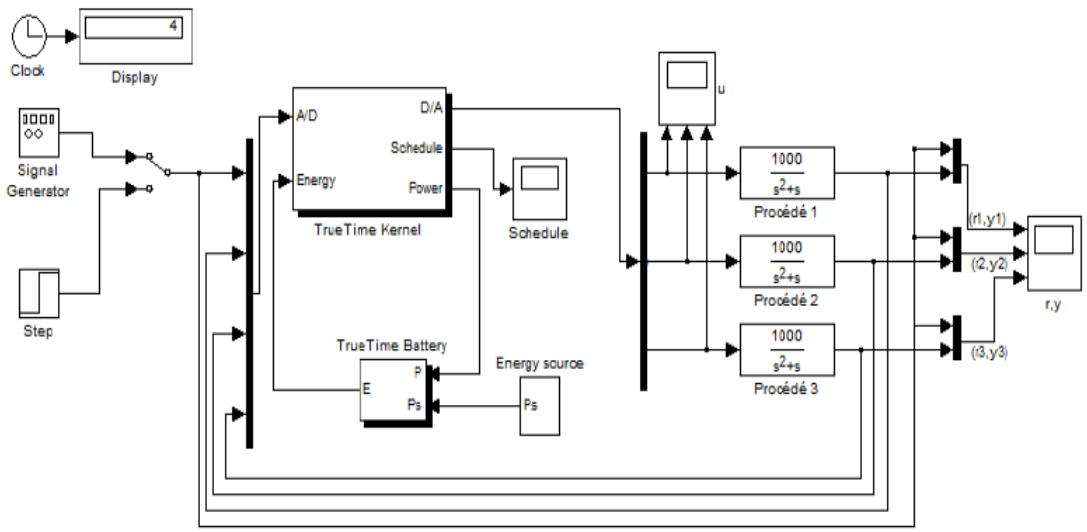

FIGURE 4.6 – Schéma du système de contrôle

4.7.2 Résultats de l'évaluation

Nous évaluons, par simulation, les performances de l'algorithme FBS-resources et nous montrons l'effet du choix de la valeur du seuil de la batterie sous l'algorithme FBS-resources avec la supposition que les tâches s'exécutent avec leurs WCET puis avec leurs durées d'exécution actuelles (c_i), en termes de consommation d'énergie, et de facture de vitesse du processeur.

1- Ordonnancement régulé des tâches sous leur WCET

La figure 4.7 montre que la tâche 1, est la plus prioritaire, a été bloqué à un instant (t=0,12 ms) par la tâche t3 qui accède à la même ressource partagée demandée par t1. Et ceci montre les résultats de fonctionnement de SRP.

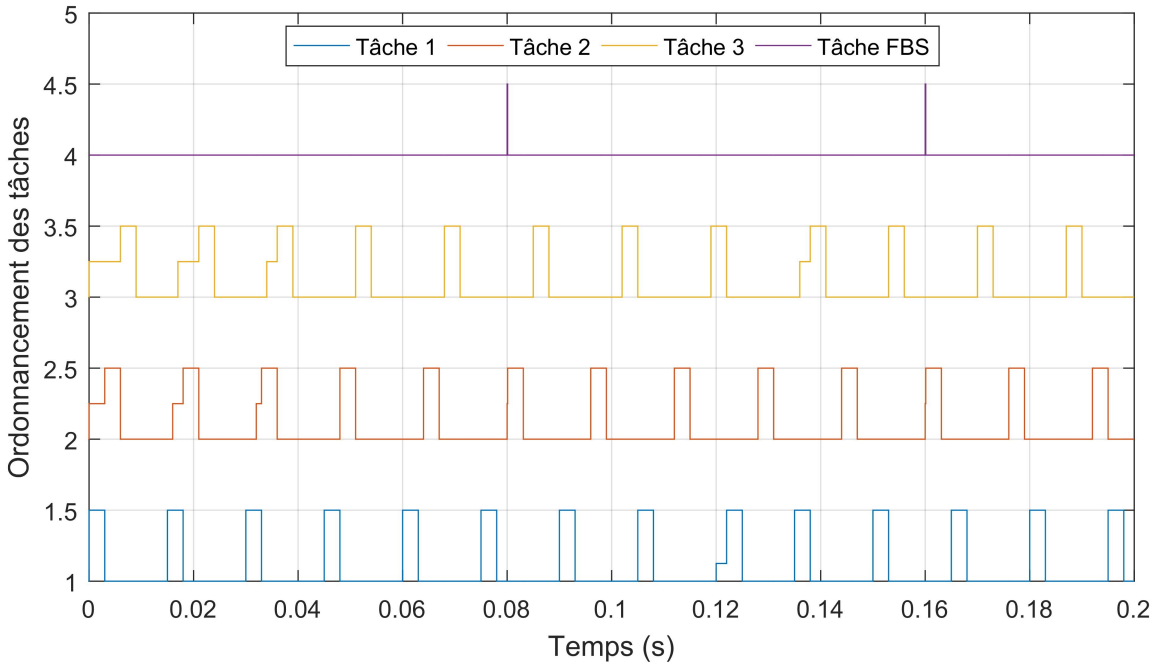

FIGURE 4.7 – Ordonnancement régulé des tâche sous leur WCET et L=2,5

2- Facture de vitesse

Nous présentons ici les résultats montrant la variation de facture de vitesse dans les trois cas suivant :

- Facteur de vitesse lorsque nous considérons WCET et L=2.5 (seuil de la batterie) La figure 4.8 montre une variation rapide de facture de vitesse du processeur et ces changements sont compris entre S =0.8 et S = 1
- Facture de vitesse sans considération de WCET et L= 0,5 La figure 4.9 montre une variation très rapide de facture de vitesse du processeur et ces changement sont compris entre S =0.4 et S =0.6. Ceci implique moins de consommation d'énergie (voir figure 4.12)
- Facture de vitesse sans considération de WCET et L= 2,5 La figure 4.10 montre peu de changements du facture de vitesse comparativement au cas précédant.ces changement sont compris entre S =0.8 et S =1

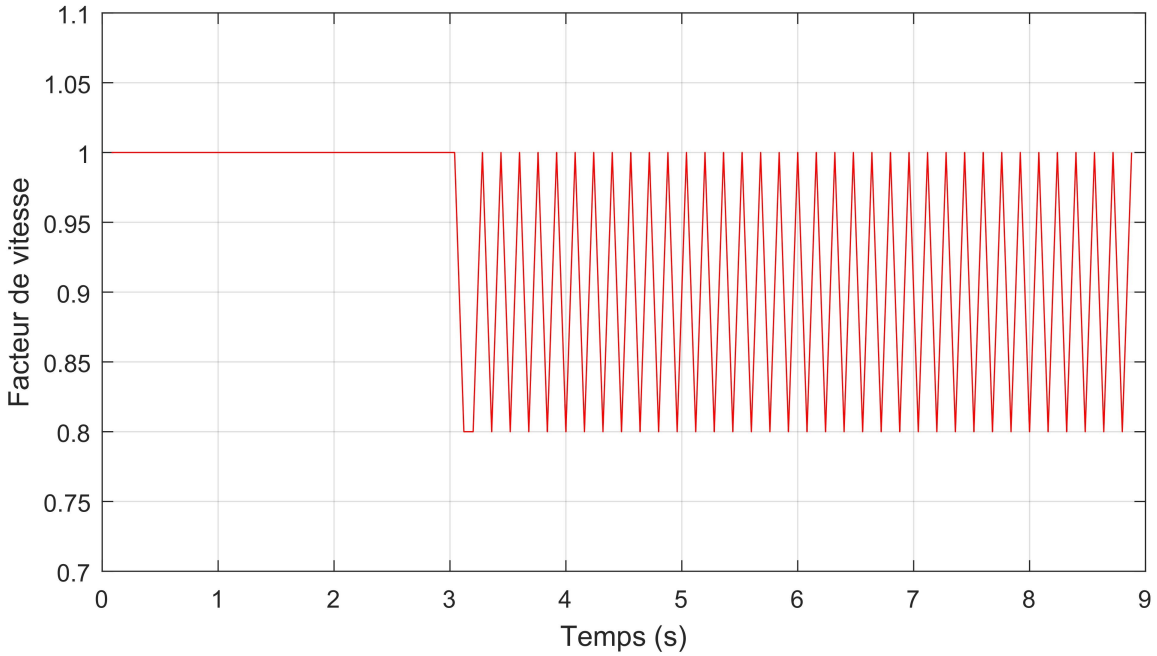

FIGURE 4.8 – Facteur de Vitesse avec L 2,5 avec WCET

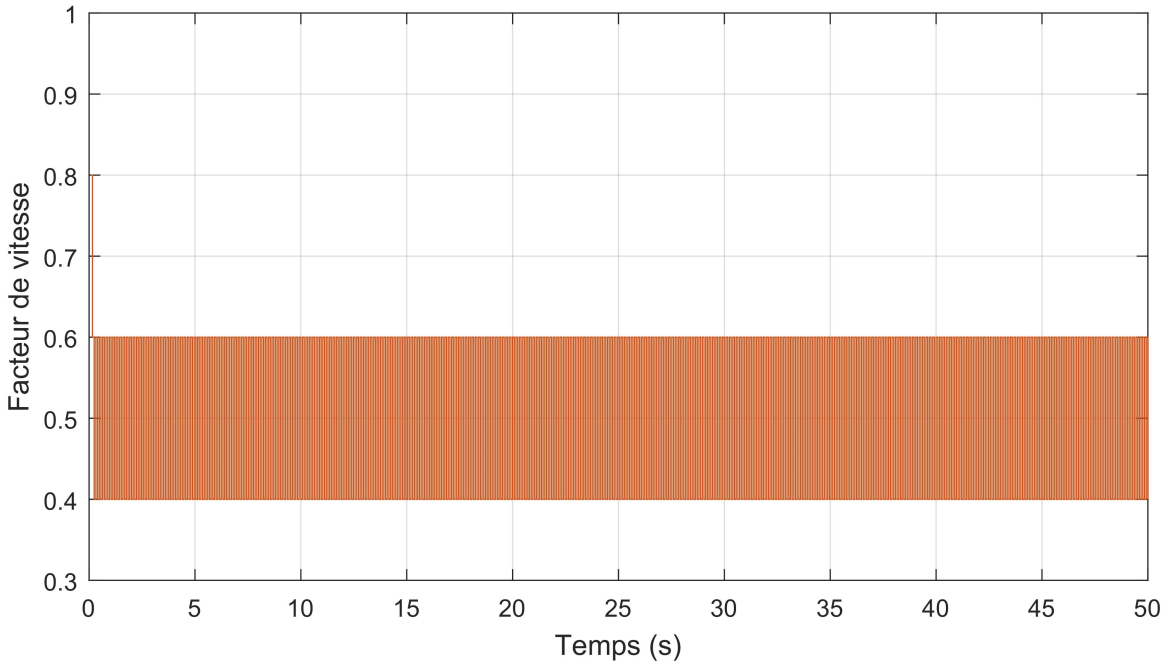

FIGURE 4.9 – Facteur de Vitesse avec L 0,5 sans WCET

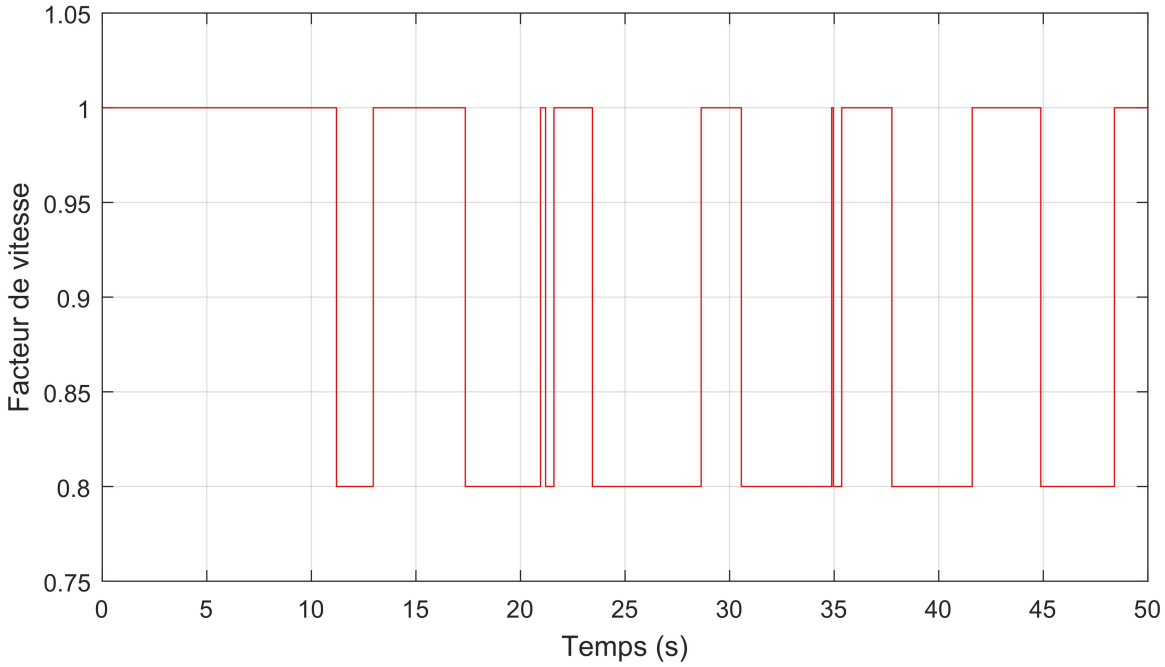

FIGURE 4.10 – Facteur de Vitesse avec L 2,5 sans WCET

3- Énergie disponible

La figure 4.11 montre l'énergie disponible sous FBS-resources lorsque L=0,5 et L= 2,5 où les tâches s'exécutent avec leurs WCET. La figure 4.12 montre l'énergie disponible sous FBS-resources avec L=0,5 et L= 2,5 où les tâches s'exécutent avec leurs durée actuelle. Notons que le temps de simulation 50_s. Les résultats de la figure 4.11 montre que la batterie se décharge après un certain temps dans les deux cas L=0,5 et L=2,5. Ceci cause l'arrête de l'exécution et la déstabilisation des procédés contrôlés. Et notons que la figures 4.12 présente la plus faible consommation d'énergie où l'énergie minimale disponible avec L= 0,5 est égale à E_{min} = 1,8 Joule, et avec L= 2,5 est E_{min}=1,7. Et donc FBS-resource dans ce cas protège contre la décharge de la batterie.

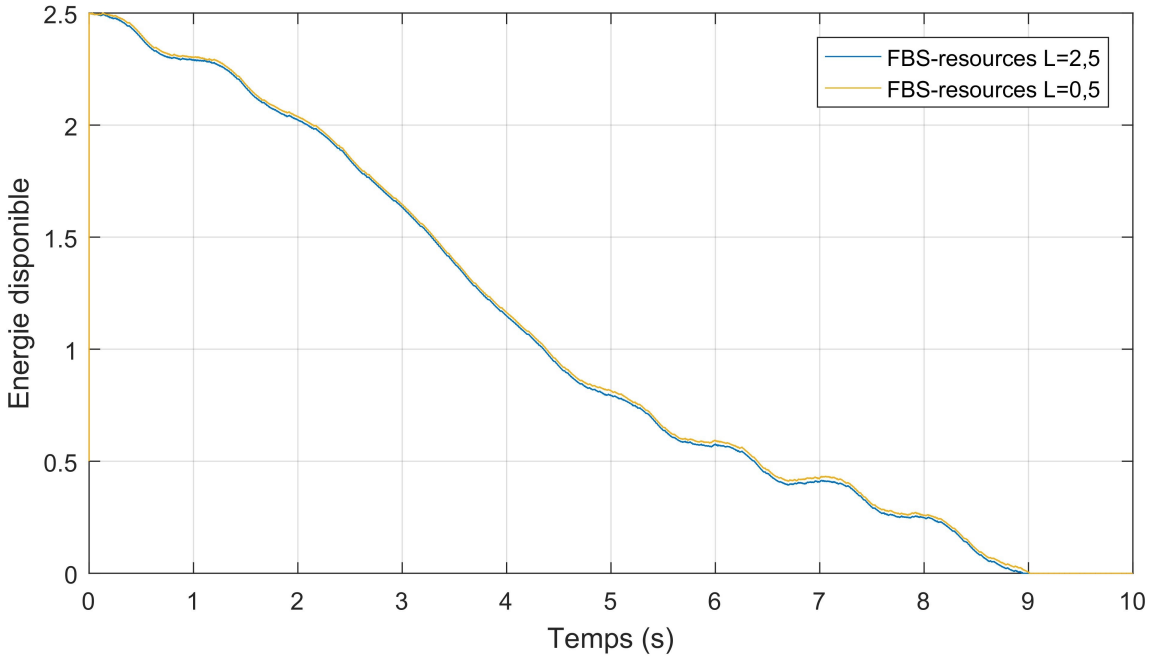

FIGURE 4.11 – l'énergie disponible avec FBS ressources L2-5 et L0-5 avec WCET

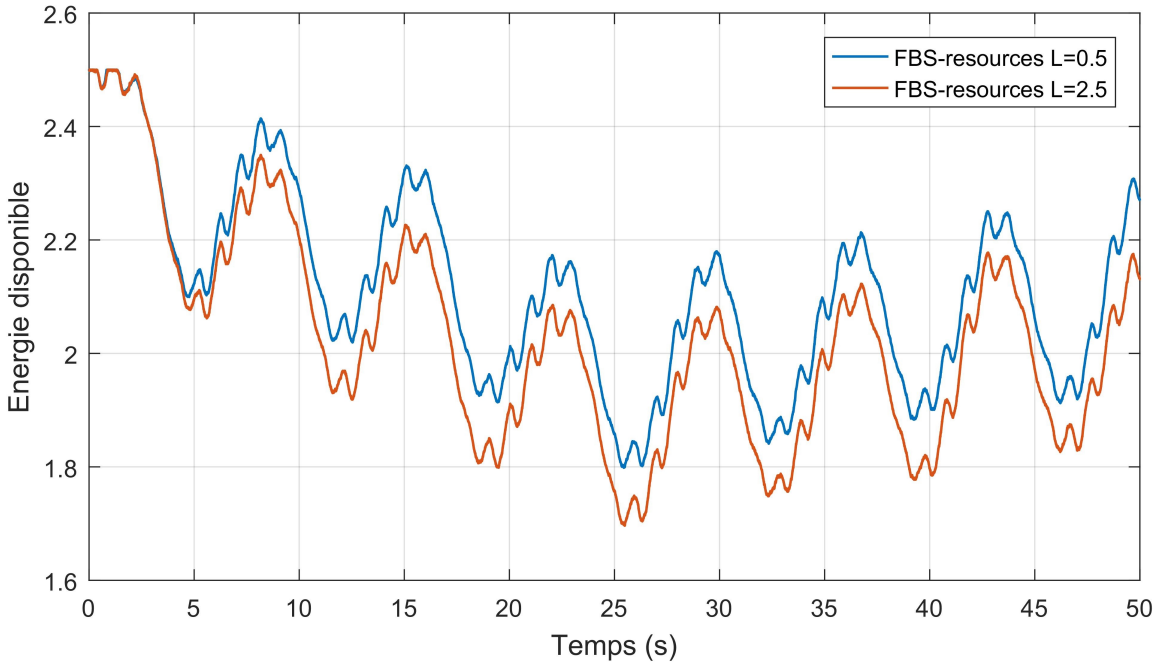

FIGURE 4.12 – l'énergie disponible avec FBS ressources L2-5 et L0-5 sans WCET

4.8 Conclusion

Dans ce chapitre, nous avons visé l'intégration de la commande par rétroaction dans l'ordonnancement temps réel des tâches périodiques sous contraintes de ressources critiques dans les systèmes de récupération d'énergie renouvelable.

Nous avons présenté un algorithme régulateur d'ordonnancement, applicable aux systèmes temps réel qui récupère l'énergie ambiante pour assurer son autonomie énergétique. La solution que nous avons proposées visent à prendre en considération le partage des ressources par les tâches et les incertitudes des durées d'exécution, et la quantité d'énergie disponible dans la batterie afin d'étendre la durée de vie des batteries.

Les résultats obtenus montrent que la solution proposée étend la durée de vie des batteries du système considéré.

Conclusion générale

L'objectif de ce mémoire est l'étude du problème d'ordonnancement régulé des tâches périodiques sous contraintes des ressources et d'énergie renouvelable.
Pour cette étude, nous avons consacré une partie de notre travail à l'état de l'art, qui constitué de trois chapitres.
Dans le premier chapitre, généralité sur les systèmes temps réel, dans laquelle nous avons introduit le problème de l'ordonnancement temps réel, ainsi que la modélisation des tâches temps réel. Ensuite, nous avons présenté les différentes approches et algorithmes existant dans la littérature traitant les problèmes d'ordonnancement temps réel. Cette présentation nous permettant d'améliorer nos connaissances dans le domaine de l'ordonnancement temps réel afin de mieux appréhender notre étude.

Dans le deuxième chapitre, nous avons présenté les stratégies de commande des procédés. Ensuite, nous avons donné les paramètres, les critères de performances et les méthodes d'évaluation des systèmes de commande. Puis, nous avons abordé la problématique d'ordonnancement basé sur WCET. Nous avons parlé, par la suite, de l'ordonnancement temps réel régulé et des approches proposées dans la littérature pour l'adaptation du principe de la commande par rétroaction à l'ordonnancement temps réel. Nous avons vu que ces approches permettent de tenir compte de l'évolution imprévisible des durées d'exécution des tâches dans le système et de s'y adapter.

Dans le troisième chapitre, nous avons présenté le problème et les solutions proposées en rapport avec l'ordonnancement temps réel sous contrainte de la consommation d'énergie du processeur. Nous avons constaté que les solutions proposées sous contraintes d'énergie ne prennent pas en considération l'ordonnancement régulé des tâches périodiques sous

contraintes de ressources, Nous avons remarqué que les approches proposées dans ces solutions considèrent les pires durées d'exécution (WCET) des tâches, ce qui induit une sous-utilisation des ressources de traitement et de communication et un surdimensionnement coûteux, surtout en énergie consommée.

Dans le quatrième chapitre (contribution), nous avons détaillé notre contribution dans l'ordonnancement temps réel régulé des tâches périodiques sous contraintes des ressources et d'énergie renouvelable ; Dans ce contexte, nous avons proposé un algorithme de calcul des facteurs de vitesse pour l'adaptation dynamique de la tension dans un contexte d'ordonnancement temps réel de tâches périodiques. Des contraintes de partage de ressource ont été étudiées dans un contexte de minimisation de l'énergie. Nous avons montré que l'algorithme de calcul de facteur de vitesse donne des résultats probants sur le plan des simulations.

Dans ce travail, nous n'avons pas comparé notre algorithme avec ceux de la littérature, car aucun d'eux ne prend en considération l'ordonnancement des tâches périodiques sous contraintes de ressources, et d'énergie renouvelable.

Ce travail nous a été très bénéfique et nous a permis d'approfondir nos connaissances dans le domaine de systèmes temps réel embarquées. Cependant que le temps imparti à préparer ce mémoire est loin d'être suffisant, mais grâce au Dieu et notre volonté et l'accompagnement de notre encadreur et sa patience nous avons pu de l'achever dans son délai critique. Ce travail nous ouvre de larges perspectives pour le compléter et l'étendre dans d'autres dimensions. Par exemple, nous planifions de développer des algorithmes pour l'ordonnancement temps réel régulé des tâches périodiques et apériodiques sous contraintes de ressources, de précédence et d'énergie renouvelable et d'étendre la problématique restreinte dans notre travail à une architecture monoprocesseur, aux applications distribuées.

Bibliographie

[1] ABBAS Akli. *Application de Techniques de Commande par Rétroaction pour l'ordonnancement Temps Réel des Tâches dans les Systèmes Industriels.* PhD thesis, École nationale Supérieure d'Informatique, 2016.

[2] Sébastien Gérard Jérôme Hugues Yassine Ouhammou Sara Tucci-Piergiovanni Francis Cottet, Emmanuel Grolleau. *SYSTÈMES TEMPS RÉEL EMBARQUÉS,Spécification, conception, implémentation et validation temporelle.* Dunod, 2ème edition 2014.

[3] Emmanuel Grolleau Francis Cottet. *SYSTÈMES TEMPS RÉEL DE CONTRÔLE COMMANDE,Conception et implémentation.* Dunod, 2005.

[4] Hui ZHANG. *Gestion de l'énergie renouvelable et ordonnancement temps réel dans les systèmes embarqués.* PhD thesis, UNIVERSITÉ DE NANTES FACULTÉ DES SCIENCES ET DES TECHNIQUES, 2012.

[5] Maissa Abdallah. *Ordonnancement temps réel pour l'optimisation de la Qualité de Service dans les systèmes autonomes en energie.* PhD thesis, Université de Nantes, 2014.

[6] Emmanuel GROLLEAU. *Ordonnancement dans les systèmes temps réel Chapitre Introduction à l'ordonnancement temps réel.* ISTE Editions, 2014.

[7] Ahmed RAHNI. *Contributions à la validation d'ordonnancement temps réel en présence de transactions sous priorités fixes et EDF.* PhD thesis, Université de Poitiers, 2008.

[8] Bruno Sadeg. *Systèmes temps réel et Ordonnancement.* 2012.

[9] M. Chetto-M. Silly-T. Bouchentouf. *Dynamic scheduling of real-time tasks under precedence constraints.* J. of Real-Time Systems, 2, pp. 181-194, 1990.

[10] Daniel Simon. *Conception conjointe commande/ordonnancement et ordonnancement réegulé.* 2005.

[11] HAMID Rabah. *Ordonnancement Temps Réel des Tâches dans un Système Industriel par Intégration d'une loi de Régulation Automatique.* PhD thesis, Université de M'Sila, 2013.

[12] MARYLINE Chetto. *Ordonnancement dans les systèmes temps réel.* ISTE Editions, 2014.

[13] Patrick Andrianiaina. *Commande robuste avec relâchement des contraintes temps-réel.* PhD thesis, Université de Grenoble, 2012.

[14] David Robert. *Contribution à l'interaction commande/ordonnancement.* PhD thesis, Institut National Polytechnique de Grenoble, 2007.

[15] Daniel SIMON-Ye-Qiong SONG et Olivier SENAME. *Ordonnancement dans les systèmes temps réel Chapitre Conception conjointe commande-ordonnancement.* ISTE Editions, 2014.

[16] Per ; Årzén Karl-Erik Eker, Johan-Hagander. *A Feedback Scheduler for Real-Time Controller Tasks .Control Engineering Practice, 8(12), 1369-1378.* 2000.

[17] Cervin Anton. *Integrated Control and Real-Time Scheduling.* PhD thesis, Lund University, 2003.

[18] www.energies-renouvelable.org date consltation 20-08-2018.

[19] Nicolas Navet Bruno Gaujal. *Ordonnancement temps réel et minimisation de la consommation d'énergie.* 2006.

[20] Frédéric Parain Michel Banâtre Gilbert Cabillic Teresa Higuera Valérie Issarny Jean-Philippe Lesot. *Techniques de réduction de la consommation dans les systèmes embarqués temps-réel.* 2000.

[21] Hanene Ben Fradj. *Optimisation de l'énergie dans une architecture mémoire multi-bancs pour des applications multi-tâches temps réel.* PhD thesis, Université de Nice-Sophia Antipolis, 2006.

[22] R Jejurikar, R. et Gupta. *Energy aware task scheduling with task synchronization for embedded real time systems.* 2002.

[23] T.P. BAKER. *Stack-Based Scheduling of Realtime Processes*. 1991.

[24] André Allavena ; Daniel Mossé. *Scheduling of Frame-based Embedded Systems with Rechargeable Batteries*. 2001.

[25] C. Moser-D. Brunelli-L. Thiele L. Benini. *Real-Time Scheduling with Regenerative Energy*. 2006.

[26] Rafic Hage Chehade Maryline Chetto, Hussein El Ghor. *Real-Time Scheduling for Energy Harvesting Sensors*. 2011.

[27] Hussein El Ghor-Maryline Chetto-Rafic Hage Chehade. *RA Real-Time Scheduling Framework for Embedded Systems with Environmental Energy Harvesting*. 2010.

[28] Shaobo Liu Qinru Qiu and Qing Wu. *Energy Aware Dynamic Voltage and Frequency Selection for Real-Time Systems with Energy Harvesting*. 2008.

[29] QingWu Shaobo Liu JunLu and Qinru Qiu. *Harvesting-Aware Power Management for Real-Time Systems With Renewable Energy. TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*. 2011.

[30] Shaobo Liu-Qing Wu and Qinru Qiu. *An Adaptive Scheduling and Voltage/Frequency Selection Algorithm for Real-time Energy Harvesting Systems*. 2009.

[31] A. Abbas M. Loudini W. K. Hidouci. Energy and quality of control constraints in real-time scheduling of synchronous hybrid tasks. 2013.

[32] ABBAS Akli-Emmanuel Grolleau-Malik Loudini-Walid-Khaled Hidouci. *A Real-Time Feedback Scheduler based on Control Error for Environmental Energy Harvesting Systems*. 2016.

[33] Akli Abbas Emmanuel Grolleau-Driss Mehdi-Malik Loudini-Walid Khaled Hidouci. *A Real-Time Feedback Scheduler for Environmental Energy with discrete Voltage/Frequency Modes*. 2014.

[34] Abbas A-Grolleau E-Loudini M. et Hidouci W.-K. *A real-time feedback scheduler based on control error for environmental energy harvesting systems*. 2015.

[35] RUIBIN XU DANIEL MOSSE and RAMI MELHEM. *Minimizing Expected Energy Consumption in Real-Time Systems through Dynamic Voltage Scaling*. 2007.

[36] Gang Chen Kai Huang Jia Huang Christian Buckl-Alois Knoll. *Effective Online Power Management with Adaptive Interplay of DVS and DPM for Embedded Real-time System*. 2013.

[37] Clemens Moser-Davide Brunelli-Lothar Thiele-Luca Benini. *Real-time scheduling for energy harvesting sensor nodes*. 2007.

[38] http ://www.control.lth.se/truetime/ ;date consultation 18-09-2018.