République Algérienne Démocratique et Populaire
Ministère d'Enseignement Supérieur et de la Recherche Scientifique
UNIVERSITE ABBAS LAGHROUR KENCHELA
FACULTE DES SCIENCES ET DE LA TECHNOLOGIE
Département de mathématique et informatique

Mémoire de fin d'études

Pour obtenir le diplôme de Master (LMD)
Spécialité : Informatique
Option : Génie Logiciel et Systèmes Distribués

Optimisation de la consommation d'énergie sous contraintes Temporelles au niveau des systèmes embarqués distribués temps Réel.

Encadré Par :

MEHALAINE Ridha

Présenté Par :

KHALFAOUI Souha

BELGROUNE Nassir

Année Universitaire 2020-2021

A mes chers parents

Remerciements :

D’abord. Louage à dieu qui m’a donnée la force et la santé pour contenue.

Te tiens à remercier mon encadreur **Dr. MEHALAINE** professeur d'informatique à l’université **d’ABBAS LAGHROUR - KHENCHELA**, pour son aide. Ses conseils et sa patience. Sans lui, ce travail ne verra pas le jour.

Je tiens à remercier les membres de jury m’avoir accordé cet honneur d’être devant eux.

Un remerciement particulier pour tous ceux qui m’ont aidé de près ou de loin à la réalisation de ce travail.

Enfin, mes derniers et plus forts remerciements vont à mes parents qui ont toujours cru en moi, qui m’ont soutenu moralement tout au long de mes études. J’ai conscience des efforts qu’ils ont faits pour moi et je ne saurais jamais les remercier assez de cela.

Résumé :

Cette recherche vise à répondre aux besoins des nouvelles applications temps-réel dans les systèmes embarques, pour lesquelles la consommation d'énergie joue un rôle très important.

Pour cela, Nous présentons dans ce travail, notre solution qui consiste à fournir un modèle d'ordonnancement des tâches périodique indépendantes sur une architecture multiprocesseur sous contraintes temporelles visant à minimiser la consommation d'énergie par l'utilisation de processeur avec la vitesse minimale et qui respect les échéances des taches, par l'application de l'algorithme EDF et la technique DVS. Nous avons simulé notre solution en C++ pour leur facilité et efficacité.

Ce mémoire traite uniquement l'ordonnancement des tâches périodique indépendantes sur une multiprocesseur sous contraintes temporelles, les travaux futurs vont s'orienter vers la conception d'ordonnanceurs des tâches périodiques et apériodique/ sporadiques avec dépendance répondant aux mêmes besoins. En effet, celles-ci sont de plus en plus employées dans un grand nombre des systèmes embarqués, non seulement dans le domaine du traitement d'information, mais également pour le contrôle de robots et la simulation temps réel de systèmes dynamiques, du fait qu'ils procurent une excellente efficacité de coût.

Mots clés: Systèmes Embarqués, Systèmes temps réel, Systèmes distribuée, technique DVS, Consommation d'énergie, EDF.

Abstract:

This research aims to meet the needs of new real-time applications in real-time systems, where energy consumption plays a very important role.

For this, we present in this work, our solution which consists in providing a scheduling model of independent periodic tasks on a multiprocessor architecture under time constraints aiming to minimize the energy consumption by the use of processor with the minimum speed and which respects the deadlines of the tasks, by applying the EDF algorithm and the DVS technique. We simulated in C ++ for their ease and efficiency.

This dissertation dealing only scheduling independent periodic tasks on a multiprocessor

architecture under time constraints, future work will be directed towards schedulers design periodicals and aperiodic tasks / sporadic arm meeting the same needs. Indeed, they are increasingly used in many embedded systems, not only in the field of information processing, but also for robot control and real-time simulation of Dynamic systems, the fact that they provide excellent cost effectiveness.

Keywords: Embedded Systems, Real Time Systems, distributed systems, DVS technology, Energy consumption, EDF.

ملخص :

يهدف هذا البحث إلى تلبية احتياجات تطبيقات الوقت الحقيقي الجديدة في أنظمة الوقت الحقيقي الحديثة، يلعب استهلاك الطاقة دورًا مهمًا للغاية
لذلك ، نقدم في هذا العمل حلنا الذي يتمثل في توفير نموذج جدولة للمهام الدورية المستقلة على بنية متعددة المعالجات في ظل قيود زمنية تهدف إلى تقليل استهلاك الطاقة عن طريق استخدام المعالج بأقل سرعة والتي تحترم المواعيد النهائية لـ المهام ، من خلال لسهولة و كفاءة C ++ تطبيق خوارزمية "الأولوية لأقرب نهاية الوعد" وتقنية نظام التوتر المتغير. قمنا بمحاكاة.

تتناول هذه المذكرة جدولة المهام الدورية والمستقلة في وظيفة واحدة أو وظائف متعددة في ظل قيود منتظمة، وسيتم معالجة الخطأ في المستقبل لمعالجة الاهتمامات الدورية أو غير الدورية ، مستقلة أو مستقلة ، صلبة أو مرنة ، والتي تعتمد على نهائية تحقيق الهد .نظام مرن واحد في واحد، ليس فقط لمعالجة القضيب، ولكن أيضًا لمراقبة الآليات والمحاكاة في الوقت الفعلي للنظام
صحية ، لأن أنظمتها توفر ربحية ممتازة

الكلمات المفتاحية: الأنظمة المدمجة ، أنظمة الوقت الحقيقي ، الأنظمة الموزعة ، تكنولوجيا نظام التوتر المتغير ، استهلاك الطاقة ." ، خوارزمية "الأولوية لأقرب نهاية الوعد

Table des matières :

Table des figures:

Introduction Générale

Introduction générale

1. Contexte du mémoire

Les systèmes embarqués (SE) sont de plus en plus présents dans notre quotidien. Cette technologie peut être trouvée aussi bien dans les systèmes multimédia que dans les applications de réseau et de télécommunication. L'utilisation des SE est en train de suivre une courbe exponentielle et on attend même que leurs ventes dépassent en revenus celles des processeurs de PC à usage général.

Un SE est un système à application spécifique qui contient du matériel et du logiciel adapté à une tâche particulière et qui fait partie d'un système plus large qui n'est pas forcement informatique (ex. électronique, mécanique, etc.). Un SE interagit avec le monde extérieur via les capteurs/actionneurs et soumis à des contraintes spatiales, temporelles et énergétiques très strictes. La conception de tels systèmes complexes face plusieurs défis et nécessitent une collaboration entre plusieurs équipes appartenant à des disciplines différentes (ex. logiciel, matériel, système, intégrateur, etc.). Le domaine des systèmes embarqués est donc pluridisciplinaire.

En effet, SE sont de nature hétérogène du fait de l'hétérogénéité des applications qu'ils implémentent. Ils combinent typiquement des parties logicielles (des processeurs à usage général, des processeurs à usage spécifique, des processeurs de traitement du signal et d'autres matérielles.

En fin ces systèmes sont de plus en plus souvent réalisés avec des architectures multiprocesseurs (à savoir distribuées), c'est-à-dire comportant plusieurs processeurs. Ceci est dû d'une part au besoin de puissance de calcul qu'un seul processeur (monoprocesseur) ne peut fournir et d'autre part à un besoin de modularité et de flexibilité, à la fois lors de la conception système pour réutiliser et faire évoluer le nombre de processeurs utilisés et à la fois pour rapprocher les capteurs et les actionneurs des processeurs qui les traitent.

2. Problématique et motivations

Un SE est dit temps réel s'il est capable de satisfaire ses contraintes temporelles. En fait, nous pouvons classer les SE selon le type de contrainte de temps en trois classes : les SE durs, souples et fermes.

Dans ce qui suit, nous nous sommes intéressés par les SE temps réel dirigés par le temps. Depuis que ces systèmes disposent des batteries autonomes, leur conception doit minimiser la consommation énergétique afin de prolonger la durée de vie de ces batteries. La gestion de la consommation d'énergie est devenue donc un enjeu majeur dans tels systèmes. Parmi les solutions envisageables, on trouve notamment les algorithmes d'ordonnancement temps réel qui

prennent en considération l'optimisation de la consommation d'énergie

La problématique qui se pose au niveau des SE est comment concevoir des systèmes qui satisfont à la fois les contraintes temporelles et énergétiques sachons que ces deux objectifs sont antagonistes. Pour remédier à ce dilemme, les spécialistes du domaine ont développé une nouvelle classe des algorithmes d'ordonnancement temps réel dite algorithmes d'ordonnancement prenant en compte la réduction de la consommation d'énergie 'energy aware scheduling algorithms'. Ces algorithmes combinent entre les algorithmes d'ordonnancement temps réel et quelques techniques connues de la gestion de la consommation énergétique au niveau des processeurs embarqués. Une autre tendance est de considérer ce problème comme étant un problème d'optimisation combinatoire puis d'appliquer les méthodes d'optimisation combinatoire exactes comme la programmation linéaire ou les heuristiques/métaheuristiques comme l'algorithme EDF pour minimiser la consommation d'énergie.

Une analyse approfondie du problème d'ordonnancement temps réel qui prend en compte la minimisation de la consommation d'énergie, nous a conduits à l'aide de la technique DVS. Ce constat est justifié par le fait qu'on traite des informations imprécises et incertaines. Ces informations englobent par exemple les temps d'arrivées des tâches, les temps d'exécutions réels des tâches qui sont généralement très loin de leurs temps d'exécution au pire de cas, les priorités des taches, la meilleure fréquence d'horloge du processeur qui mène à la consommation minimale, le bon moment pour migrer une tâche vers un autre processeur, etc. L'incertitude intrinsèque dans les systèmes temps réel et en particulier les systèmes dynamiques augmente les difficultés d'algorithmes d'ordonnancement conventionnels pour optimiser la consommation d'énergie.

3. Objectifs et contribution

L'objectif de ce travail est d'appliquer l'algorithme EDF et la technique DVS pour minimiser la consommation d'énergie dans les systèmes embarqués temps réel à tâches périodiques indépendantes sur une architecture multiprocesseurs.

L'avantage de la technique DVS est pour aider l'ordonnanceur à prendre les bonnes décisions au bons moments le plus tôt possible pour choisir les priorités des tâches les plus appropriées pour minimiser la consommation d'énergie globale du système tout en respectant les échéances des tâches et maintenir les taux d'utilisation des processeurs aux niveaux élevés. Les objectifs à minimiser sont : l'énergie, le temps moyen de réponse.

4. Organisation du mémoire

Notre mémoire est organisé comme suit :

Le premier chapitre est divisée en deux parties : la première partie est une introduction au

système l'embarqué. La deuxième partie détaille l'ordonnancement temps réel et donne une taxonomie des différents algorithmes d'ordonnancement temps réel.

Dans le deuxième chapitre, nous dresserons les problématiques de l'estimation et les techniques de réduction de la consommation d'énergie. Il présente également un état de l'art et les travaux voisins portant sur l'ordonnancement temps réel et la réduction de la consommation d'énergie.

Le dernier chapitre présente notre contribution (conception et implémentation) de de l'approche proposée avec un exemple explicatif.

Enfin, nous concluons ce mémoire en présentant les apports de notre solution et en dégageant les perspectives.

Chapitre 1

Systèmes embarqués et Systèmes temps réel

Systèmes Embarqués et Systèmes temps réel

1. Introduction

Aujourd'hui, les domaines de l'informatique prennent une place de plus en plus importante dans notre société. Parmi ceux-ci figurent les systèmes embarqués en temps réel. On les utilise souvent quotidiennement sans même qu'on ne s'en rende compte. Ils servent à contrôler presque tous les appareils électroniques qu'on connaît aujourd'hui, on les retrouve vraiment partout, dans les véhicules, smart watches, jeux, téléviseurs, machines électroménagères, ordinateurs …

Ces systèmes embarqués qui considéré comme un domaine vaste à l'étude soit dans l'étape de conception, implémentation ou validation…

Ce chapitre vise à décrire le contexte de notre étude, qui est le domaine des systèmes embarqués distribués sous les contraintes de temps réel. Dans un premier temps nous présentons les systèmes embarqués vises dans l'étude dont nous identifions les caractéristiques et les éléments de constitution qui nous intéressent. Ensuite, nous nous focalisons sur les systèmes temps réel et les différents algorithmes d'ordonnancement les plus connus.

I. Systèmes embarqués

I.1.Définition :

Un système embarqué peut être défini comme un système électronique et informatique autonome, souvent temps réel, spécialisé dans une tâche bien précise. Ses ressources sont généralement limitées. [MES 19]

On le définit aussi généralement par le fait qu'il n'est pas visible en tant que tel, mais est intégré dans un équipement doté d'une autre fonction, on dit aussi que le système est enfoui.

Il ne possède généralement pas des entrées/sorties standards et classiques comme un clavier ou un écran d'ordinateur. Le terme désigne aussi la combinaison entre le matériel et le logiciel, c'est-à-dire le matériel et l'application sont intimement liés et noyés dans le matériel, et ne sont pas facilement discernables comme dans un environnement de travail classique de type ordinateur PC. [KAD 05]

Typiquement, le SE est un système sur un ou plusieurs processeurs et dont les programmes sont stockés en ROM. A priori, tous les systèmes qui ont des interfaces digitales (i.e. montre, caméra, voiture…) peuvent être considérés comme des SE. Certains SE ont un système d'exploitation et d'autres non car toute leur logique peut être implantée en un seul programme. [MEH 14]

I.2. Historique:

Le concept des systèmes embarqués a vu le jour suite à l'apparition Apollo Guidance computer en 1967 et en 1971. Ils ont connu encore beaucoup plus d'envergure avec l'invention d'Intel 4004 qui était considérée comme le premier microprocesseur. A la base ces systèmes contenaient qu'un seul processeur mais actuellement ils peuvent même contenir des centaines.[OPT 19]

- 1967 : premier système embarqué de guidage lors de la mission lunaire Apollo ;
- 1971 : premier processeur produit par Intel ;
- 1984 : sortie de l'Airbus 320, premier avion équipé de commandes électriques informatisées ;
- 1998 : mise en service du métro informatisé sans conducteur Météor (ligne 14 à Paris) ;
- 1999 : introduction de l'expression « internet des objets » par Kevin Ashton ;
- 2007 : arrivée du Smartphone. [ZUI 16]

I.3. Caractéristiques d'un système embarqué :

Les systèmes embarqués se diffèrent aux systèmes traditionnels, en fonctionnement, en composants, …, et ses différences peuvent donner aux systèmes embarqués quelques caractéristiques, qui sont:

- **Autonomes** :

Le logiciel et le matériel sont considérés comme une unité unitaire, donc l'application n'est pas accessibles.

- **Les systèmes embarqués fonctionnent généralement en Temps Réel (TR)**:

Les opérations de calcul sont alors faites en réponse à un événement extérieur (interruption matérielle). La validité et la pertinence d'un résultat ne dépend pas seulement du l'exactitude des résultats, mais aussi du moment où il est délivré. Une échéance manquée induit à une erreur de fonctionnement qui peut entraîner soit une panne du système (plantage) ; soit une dégradation non dramatique de ses performances.

- **Faible encombrement, poids et consommation** :

Dans les systèmes embarqués autonomes, la consommation d'énergie est un point critique pour le coût. En effet, une consommation excessive augmente le prix de revient du système embarqué, car il faut alors des batteries de forte capacité 8 heures et plus (PC portable : 2 heures).

- **Ouverture au changement (l'environnement) :**

Les systèmes embarqués évoluent généralement dans des conditions environnementales non déterministes et souvent non maîtrisées. Ils sont exposés à des variations et autres contraintes

environnementales susceptibles d'induire des défaillances : vibrations, chocs, variation de température, variations d'alimentation, humidité, … D'où la nécessité de prendre en compte des évolutions des caractéristiques des composants en fonction des conditions environnementales.

- **Sûreté de fonctionnement (systèmes critiques) :**

Le système doit toujours fonctionner correctement, sûreté de fonctionnement du logiciel, sûreté à faible coût avec une redondance minimale, système opérationnel même quand un composant électronique lâche.

- **Faible coût :**

Optimisation du prix de revient. [BOU 12] [KAD 05]

I.4. Architecture :

Le fonctionnement du SE se résume ainsi :

Il reçoit des informations de l'environnement extérieur (monde réel) qu'il converti en signal numérique par le moyen de capteur, par exemple : Capteur de température, vitesse ...

L'unité de traitement composée du CPU, de la mémoire, du logiciel, de l'ASIC et éventuellement de système externes traite l'information avec des algorithmes.

Le traitement génère éventuellement une sortie qui est envoyée vers la sortie, les systèmes auxiliaire, les ports de monitoring ou l'IHM par le moyen d'actionneurs par exemple: vanne, moteur …

Quand un système informatique embarqué agit sur un phénomène réel (par un actionneur) dont l'évolution est mesurée (par un capteur), alors le système informatique peut contrôler ce phénomène par exemple: Un régulateur de vitesse. [UGE 19] [ZUI 16]

Il existe différents classements de l'architecture multiprocesseur. On peut les classer en :

- **(homogène/hétérogène) :** Par rapport à la nature des processeurs dont dispose l'architecture :
  - **Homogène :** Dans ce cas les processeurs sont identiques c'est-à-dire ils sont interchangeables et ils ont la même capacité de calcul.
  - **Hétérogène :** Les processeurs sont soit indépendants c'est-à-dire les processeurs ne sont pas destinés à exécuter les mêmes tâches, ou les processeurs exécutent les mêmes tâches mais chaque processeur à sa propre capacité de calcul.
- **(parallèle/distribuée) :** Selon le type de mémoire dont dispose l'architecture :
  - **Parallèle :** Un ensemble de processeurs communicant par mémoire partagée.
  - **Distribuée :** Un ensemble de processeurs à mémoire distribuée communicant par messages. [BOU 12][KER 09]

I.5. Classification des systèmes embarqués :

Les systèmes embarqués peuvent classifier selon son manière de fonctionnement en trois classes suivantes :

I.5.1 Système Transformationnel :

Activité de calcul, qui lit ses données et ses entrées lors de son démarrage, qui fournit ses sorties, puis meurt.

I.5.2 Système en interaction :

Quasi permanente avec son environnement, y compris après l'initialisation du système; la réaction du système est déterminée par les événements reçus et par l'état courant (fonction des événements et des réactions passés); le rythme de l'interaction est déterminé par le système et non par l'environnement.

I.5.3 Système Réactif ou Temps Réel :

Système en interaction permanente avec son environnement, y compris après l'initialisation du système; la réaction du système est déterminée par les événements reçus et par l'état courant; mais le rythme de l'interaction est déterminé par l'environnement et non par le système. [BOU 12] [COG 19]

I.6. Domaine d'application des systèmes embarqués:

Les systèmes embarqués aujourd'hui couvrent presque tous les domaines de notre vie quotidienne. Les domaines dans lesquels on trouve des systèmes embarqués sont de plus en plus nombreux :

- **Transports :**

Automobile : plus de 40% de sa valeur vient de ses systèmes embarqués contrôlant la climatisation, la boîte de vitesse, le tableau de bord, le moteur, détecteur de pluie…
Aéronautique : satellite, radar, avion, etc.

- **Secteur manufacturier et industrie :** Chaînes de production, automates, production et distribution d'électricité, réacteurs chimiques, réacteurs nucléaires, raffineries, dispositifs de sécurité, aide à la maintenance, etc.
- **Information et communication :** Imprimante, périphérique, téléphone, répondeur, fax, routeurs, téléphonie mobile, satellites, GPS, etc.
- **Santé :** Imagerie médicale, diagnostique, soins, implants, handicapés, etc.
- **Autres :** Carte à puce, distributeurs, etc.

- **Équipements électrique et électroménagers :** Télévision, lave vaisselle, lave linge, boîtier électronique. [OPT 19]

I.7. Exemple :

Il est difficile de citer tous les systèmes embarqués et voici un exemple plus détaillé: le portail automatique.

La figure1 présente un portail automatique permet aux personnes d'entrer ou de sortir de la maison ou de la résidence qui stock leur matricule dans une base de données. Il est un SE autonome grâce aux composants de la chaîne d'information (capteur, traiteur, actionneur).

On retrouve dans la première partie un capteur qu'il faut obtenir, c'est ici qu'on va découvrir le matricule de la voiture, c'est dans un détecteur de voie placé de chaque côté (B). On retrouve dans la deuxième partie un traiteur qui traite l'information avec un algorithme stocké dans le boîtier électronique (E). Dans la dernière patrie on retrouve un actionneur, c'est tout d'abord le moteur qui va ouvrir le portail (A) et la lumière (C) va indiquer et avertir qu'on ouvre le portail.

A : vérins.
B : cellule photographique
C : gyrophare ou flash
D : antenne
E : armoire de commande
F : clavier ou contact a clé

1 : câble 4*1.5 mm2
2 : câble téléphone 2*1
3 : câble coaxial

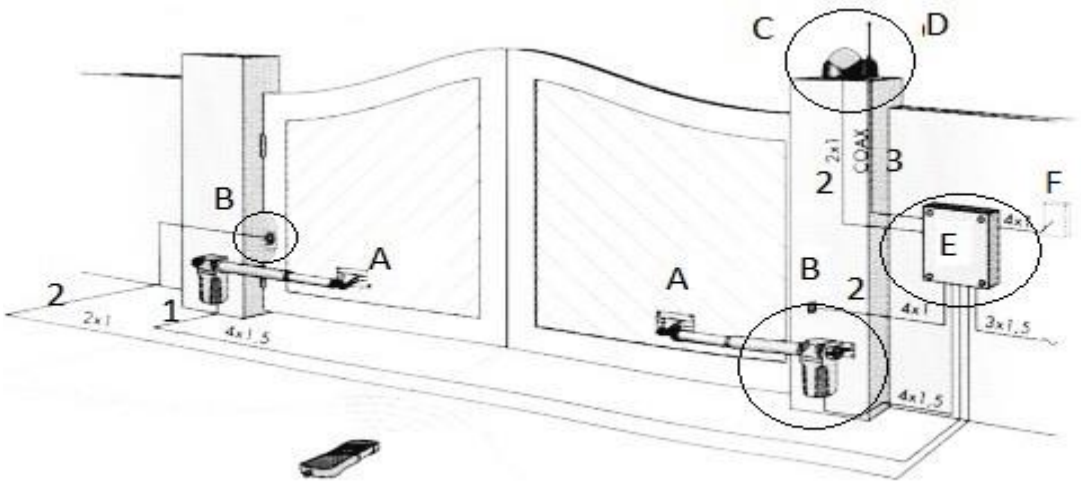

Figure 1.1 : Domaine d'application des systèmes embarqués. [TEF 14]

II. Systèmes temps réel :

II.1 Définition :

Les systèmes temps réel sont des systèmes numériques informatiques, qui implémentent une application où le respect des contraintes temporelles est la principale contrainte à satisfaire. Les systèmes temps réel se différencient des autres systèmes informatiques par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude du résultat, autrement dit le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés. Le fonctionnement est assujetti à l'évolution dynamique d'un procédé extérieur et le temps auquel les résultats sont produits non seulement de la justesse des calculs.

Un système temps réel n'est pas un système « qui va vite / rapide» mais un système qui satisfait des contraintes temporelles (les contraintes de temps dépendent de l'application et de l'environnement alors que la rapidité dépend de la technologie utilisée, celle du processeur par exemple).Donc le but n'étant pas de générer des tâches rapides, mais des tâches qui arrivent à l'heure.

Les systèmes informatiques temps réel sont aujourd'hui présents dans de nombreux secteurs d'activités : dans l'industrie de production par exemple, au travers des systèmes de contrôle de procédé (usines, centrales nucléaires), dans les salles de marché au travers du traitement des données boursières en " temps réel ", dans l'aéronautique au travers des systèmes de pilotage embarqués (avions, satellites), ou encore dans le secteur de la nouvelle économie au travers du , toujours croissant, du traitement et de l'acheminement de l'information (vidéo, données, pilotage à distance, réalité virtuelle, etc.).[BOU 12][BOU 18][PRE 00]

L'interface entre un système temps réel et son environnement et constituée par un ensemble de périphériques d'entrée et de sortie.

La figure 1.2 représente le système temps-réel contrôle un système plus large (le système de contrôle) .Les entrées appelés les capteurs qui scrutent les événements du système contrôlé fournissant des mesures, … les sorties appelés les actionneurs qui agissent sur le fonctionnement en fonction des traitements du STR sue le système contrôlé. [MAN 17]

Figure 1.2 : Définition de système temps-réel [MAN 17]

II.2 Les objectifs des systèmes temps réels :

Dans un système temps réel nous nous sommes intéressés pas seulement par des contraintes temporelles, c'est-à-dire le système doit respecter principalement les échéances, mais aussi par l'exactitude du résultat, afin du réaliser les objectifs suivants :

II.2.1 Exactitude logique (exactitude des traitements) : Calculer les bonnes sorties du système en fonction de ses entrées.

II.2.2 Exactitude temporelle : Les résultats de calcul sont présentés au bon moment (un calcul juste mais hors délai est un calcul non utilisable). En d'autres termes, un retard sur la production d'un résultat est considéré comme une erreur qui peut entraîner de graves conséquences.

II.2.3 Fiabilité : Le système répond à des contraintes de disponibilité, de temps de développement, de temps et de coûts de fabrication, d'encombrement et de poids (fiabilité du logiciel et du matériel) [BOU 12]

II.3 Classification des systèmes temps réel :

Les systèmes temps-réel peuvent se classer, selon le respect des contraintes temporelles, en trois classifications :

II.3.1 Temps réel dur ou critique (hard real time) : Lorsqu'un résultat arrivant en retard provoque un accident fatal au système (génération d'une exception).

La réponse du système dans le détail est vitale. L'absence de réponse est catastrophique (plus qu'une réponse incorrecte). La figure 1.3 illustre que une réponse dépassant son échéance est considéré comme une réponse fautive.

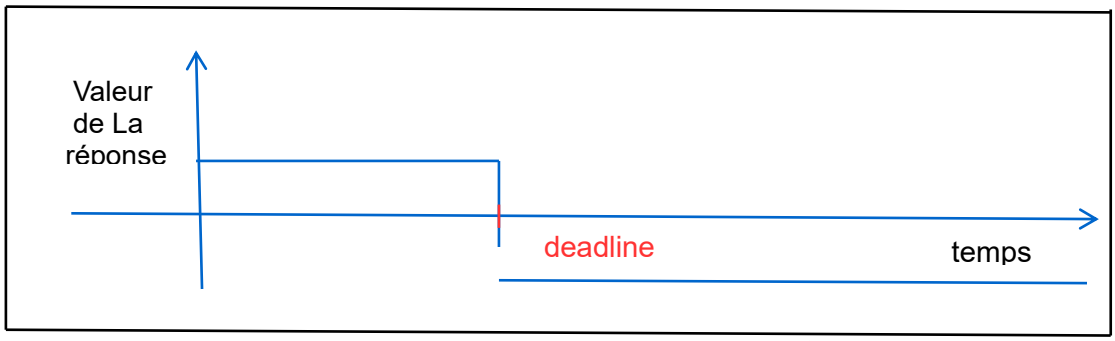

Figure 1.3 : temps réel strict [MAN 17]

Exemple: Contrôle de trafic aérien, système de conduite de missile, auto-pilotage, freinage, assistance médicalisée, …etc. [MAN 17][BOU 12]

II.3.2 Temps réel souple ou mou (soft real time) : Réduction progressive de l'intérêt d'un résultat arrivant en retard pour le système. La réponse du système après les délais réduit progressivement son intérêt. Les pénalités ne sont pas catastrophiques.

La figure 1.4 illustre qu'une réponse dépassant son échéance peut accepté à condition que la faute temporelle ne soit pas grande ne mettra pas le système en danger (erreur).

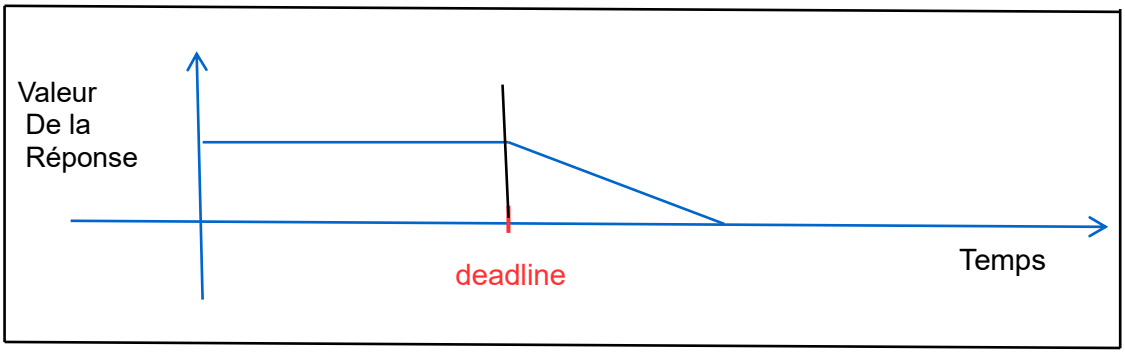

Figure 1.4 : temps réel souple ou mou. [MAN 17]

Exemple: Projection vidéo (décalage entre le son et l'image), billetterie automatique… [MAN 17][BOU 12]

II.3.3 Temps réel ferme (firm real time) : Lorsqu'un résultat arrivant en retard n'a plus d'importance au système. C'est aussi un système dans lequel le dépassement occasionnel des échéances ne met pas le système en difficulté. C'est à dire la réponse du système dans les délais est essentielle, mais ne sert plus à rien une fois le deadline passe. (Définition alternative: la pénalité de non-réponse est dans le même ordre de magnitude que la valeur de la réponse) La figure 1.5 montre que le dépassement de l'échéance n'arrête pas la tâche.

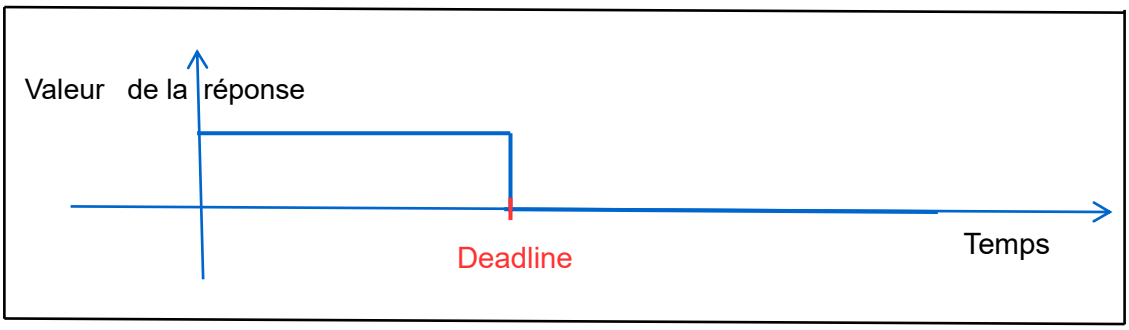

Figure 1.5 : temps réel ferme. [MAN 17]

Exemple: Projection vidéo (perte de quelques trames d'images), transactions en bourse… [MAN 17][BOU 12]

II.4 Généralités sur l'ordonnancement temps réel :

L'ordonnancement des tâches consiste à déterminer l'ordre d'allocation au processeur pour

exécuter automatiquement une tâche (une application, une commande, ...) à intervalles de temps réguliers ou bien à des moments précis. L'ordonnanceur désigne le composant du noyau du système d'exploitation choisissant l'ordre d'exécution des processus sur les processeurs d'un ordinateur. En anglais, l'ordonnanceur est appelé scheduler.

Le rôle de l'ordonnanceur du noyau, est de permettre à tous ces processus de s'exécuter à un moment ou un autre et d'utiliser au mieux le processeur pour l'utilisateur. Pour que chaque tâche s'exécute sans se préoccuper des autres et/ou aussi pour exécuter les tâches selon les contraintes imposées au système (exemple: contraintes temporelles dans le cas d'un système d'exploitation temps réel) [BOU 12].

Différentes méthodes d'implantation de l'ordonnanceur sont envisagées, selon le type de machine mono ou multiprocesseur. Pourtant, dans les deux cas, les politiques d'ordonnancement reposent sur les mêmes principes, et les informations dont elles ont besoin représentent les différents paramètres des tâches. [MEH 14]

II.5 Caractéristiques d'une tâche temps réel :

Une tâche est un ensemble d'instructions destinées à être exécutées sur un processeur. La caractérisation d'une tâche peut varier d'un modèle d'ordonnancement à l'autre et suivant la nature de celle-ci. La figure 1.6 indique un modèle de taches, parmi les paramètres les plus utilisés on peut citer :

Date d'activation R (ti) (Ready time ou Release time) : C'est la date d'arrivé de la tâche ti, elle est appelée aussi date de demande d'activation.

Début d'exécution S (ti) (Start time), fin d'exécution E (ti) (End time) : Sont respectivement la date à laquelle la tâche ti commence l'exécution sur le processeur, et la date à laquelle la tâche ti finit son exécution.

Période de tâche (T_i) : la période d'étude = [0, PPCM(Ti)].

Temps de repense RT (ti) (Response time) : Ceci représente E (ti)-R (ti).

Temps d'exécution C (ti) (Computing time) : C'est la durée d'exécution d'une tâche ti. Ce paramètre est considéré comme le pire cas des temps d'exécution (WCET pour Worst Case Execution Time) « pour les tâches Hard ». Le WCET représente une borne supérieure du temps d'exécution c'est à dire que la tâche peut se terminer plus tôt.

Echéance D (ti) (Deadline) : C'est l'échéance ou la date au plus tard. Elle représente l'instant auquel l'exécution d'une tâche doit être terminée et dont le dépassement provoque une violation de la contrainte temporelle.

Deux types de deadlines existent :

- **Relative D (ti) :** L'échéance est relative au release time de la tâche
- **Absolue D (ti) +R (ti) :** L'échéance est définie avant l'exécution

Laxité L (ti) (Laxity) : Représente le temps restant avant l'occurrence de sa date de début d'exécution ou de reprise au plus tard. Si ce paramètre vaut zéro à un instant donné, la tâche correspondante doit être impérativement exécutée à cet instant et sans interruption sinon, son échéance sera inévitablement dépassée [WIK 19].

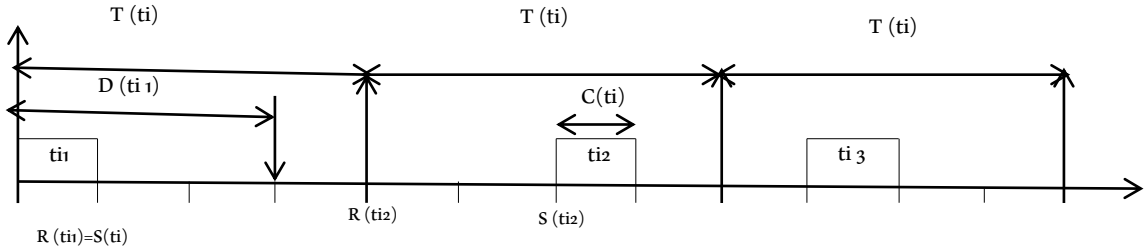

Figure 1.6 : Modèle de tâches [WIK 19]

II.6 Nature des tâches :

Nature des tâches peut être définit selon l'existence ou l'absence de différent critères, par exemple : l'inter arrivé entre les instances d'une tâche, relation entre les tâches, scénario d'exécution d'une tâche :

II.6.1 Tâches périodiques/ non périodique :

Une tâche peut s'exécuter à des intervalles réguliers (tâches périodiques) ou de manière aléatoire (tâches non périodiques).

II.6.1.1 Tâches périodiques :

On dit que la tâche ti est périodique de période T (ti) si l'événement qui conduit l'activation de cette tâche se produit à des intervalles de temps réguliers T(ti).

II.6.1.2 Tâches non périodiques :

Il existe deux types de tâches non périodiques :

II.6.1.2.1 Apériodique :

Les dates d'activations sont aléatoires et ne peuvent être anticipées. Son exécution se produit par des événements internes ou externes qui peuvent se déclencher à tout instant.

II.6.1.2.2 Sporadique :

C'est un cas particulier des tâches apériodiques où une durée de temps minimale sépare activations successives pour les prendre en compte, ces tâches sont souvent considérées comme tâchespériodiques.

II.6.2 Tâches concrètes/non-concrètes :

Si un scénario d'activation particulier est imposé aux tâches on dit qu'elles sont concrètes. Par ailleurs les tâches sont non-concrètes si les dates de première activation ne sont pas connues à

priori.

II.6.3 Synchrones/asynchrones :

Dans le cas synchrone un scénario d'activation est imposé et reproduit à l'identique pendant toute la vie du système, tandis que dans le cas asynchrone une hypothèse sur les activations des tâches est prise en compte.

II.6.4 Dépendance (précédence)/indépendance :

Les systèmes temps réel sont soit indépendantes, soit dépendantes dans le sens où une fonction produit des données pour une autre fonction qui les consomme. Une dépendance impose une précédence entre la fonction qui produit et celle qui consomme. Si on considère les tâches comme des fonctions, une contrainte de précédence entre deux tâches impose un ordre entre ces tâches. Il faut noter que cette contrainte de précédence est la plupart du temps (donc pas toujours) due au fait que les deux fonctions (tâches) sont dépendantes. Le terme « précédence » désigne un ordre entre deux tâches sans qu'il y ait transfert de données alors que « indépendance » signifie qu'un ordre n'est pas imposé entre deux tâches.

II.6.5 Hard/soft :

Si la tâche ne doit pas dépasser son échéance elle dit hard, sinon c'est-à-dire si on permet quelques fautes de dépassement temporel, on parle donc de tâche soft [LAU 18].

II.7 Classes des problèmes d'ordonnancement temps réel :

On indique ici les caractéristiques liées aux systèmes cible de l'ordonnancement ou à l'ordonnancement lui-même, qui forment les paramètres du problème d'ordonnancement qu'on cherche à résoudre.

II.7.1 Monoprocesseur/multiprocesseur : Si l'architecture ne dispose que d'un seul processeur on dit que le problème d'ordonnancement est monoprocesseur. Si plusieurs processeurs sont disponibles alors le problème est multiprocesseur.

II.7.2 Préemptif/non-préemptif : Si les tâches du système peuvent être préemptées alors le problème est préemptif. La préemption se traduit par la suspension temporaire de l'exécution d'une tâche au profit d'une autre tâche plus prioritaire.

II.7.3 En-ligne/hors-ligne (off-line/on-line): On dit qu'un ordonnancement est hors-ligne si la séquence d'ordonnancement est établie avant le lancement de l'application pour être répétée à l'infini (approche qui minimise les coûts, mais n'est pas efficace en cas de changement de l'environnement). On dit qu'un ordonnancement est en-ligne s'il se fait au fur et à mesure que les tâches arrivent dans le système, (flexible et capable de s'adapter aux changements de l'environnement, mais il est coûteux).

II.7.4 Statique/dynamique : Un ordonnancement statique (prédictif) est basé uniquement sur les paramètres des tâches au départ (avant l'activation). À l'inverse, un ordonnancement dynamique (réactif) est basé sur les paramètres des tâches qui varient au cours de l'exécution.

II.7.5 Optimal/non optimal : Un algorithme d'ordonnancement est dit optimal pour un problème donné s'il permet de trouver un ordonnancement qui respecte toutes les contraintes lorsqu'un tel ordonnancement existe. Si l'algorithme optimal ne trouve pas de solution alors aucun autre algorithme ne pourra le faire. Par ailleurs, un algorithme d'ordonnancement non-optimal vise à trouver des solutions approchées.

II.7.6 Centralisé/distribué : Lorsqu'un système est distribué, l'ordonnancement (en-ligne) est distribué si les décisions d'ordonnancement sont prises par un algorithme localement en chaque nœud. Il est centralisé lorsque l'algorithme d'ordonnancement pour tout le système, distribué ou non, est déroulé sur un nœud privilégié [PRE 00] [MAN 17] [BOU 18].

II.8 Algorithmes d'ordonnancement

Dans la suite, nous nous intéressons d'expliquer les principes des différents algorithmes d'ordonnancement sur une architecture monoprocesseur et multiprocesseur.

II.8.1 Algorithmes d'ordonnancement sur une architecture monoprocesseur

II.8.1.1 "Rate-monotonic" (RM)

L'algorithme « rate-monotonic » est un algorithme statique qui attribue la plus haute priorité à la tâche ayant la fréquence la plus élevée (la plus petite période).

La Figure 1.7 indique l'ordonnancement « rate-monotonic » de trois tâches périodiques à échéances sur requêtes : T1 (R1=0, C1=4, P1=20), T2 (R2=0, C2=1, P2=4) et T3 (R3=0, C3=2, P3=10).

La tâche la plus prioritaire est la tâche T2 (P2 =4), et la tâche la moins prioritaire est la tâche T1 (P1=20). La séquence est décrite sur l'intervalle [0,20] = [0, ppcm (Pi)], et les trois tâches respectent leurs contraintes temporelles. A observer la répétitivité de la séquence durant tout intervalle ayant la forme [k ppcm (Pi), (k +1) ppcm (Pi)], k ≥ 0, k □ N, où ppcm (Pi) représente le plus petit multiple commun des périodes Pi. [Dan04][MEH 14]

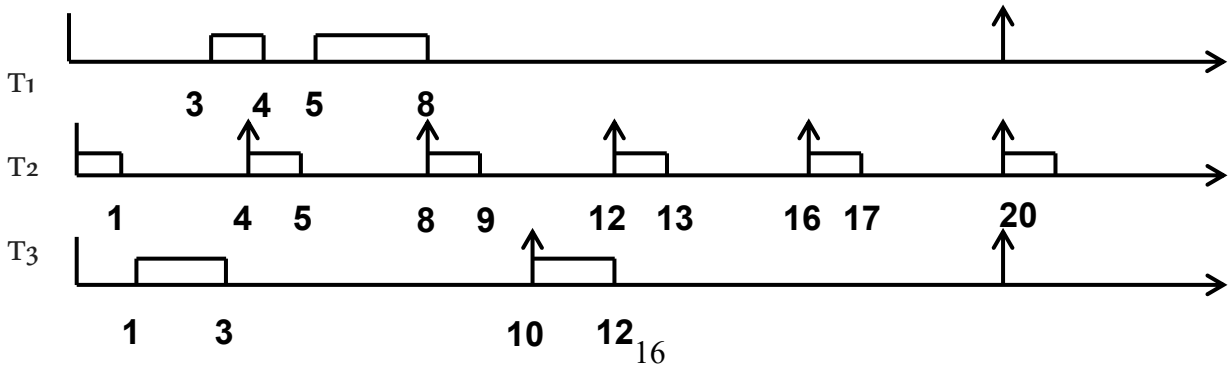

Figure 1.7: l'algorithme « rate-monotonic » [Dan04]

Remarque : Notons que l'utilisation de la périodicité comme critère d'ordonnancement limite l'applicabilité de cet algorithme aux seules tâches périodiques à échéance sur requête. L'utilisation de cet algorithme pour d'autres types de tâches ne donne aucune garantie pour le respect des échéances.

Avantages

- Simplicité de mise en œuvre.
- Optimal pour les ordonnancements à priorité statique.
- Répandu dans les exécutifs classiques.
- Bon comportement en cas de surcharge.

Inconvénients

- Surdimensionnement possible du système.
- Condition nécessaire et suffisante en O (n2).[BOU 12]

II.8.1.2 "Inverse Deadline" (ID) ou "Deadline Monotonic" (DM)

Principe de fonctionnement : l'algorithme « inverse deadline » est un algorithme statique où les priorités des tâches sont exprimées en fonction de leurs délais, la tâche la plus prioritaire étant celle qui a le plus petit délai.

La Figure 1.8 indique l'ordonnancement « inverse deadline » des trois tâches périodiques suivantes : T1 (R1=0, C1=2, D1=6, P1=10), T2 (R2=0, C2=3, D2=5, P2=10) et T3 (R3=0, C3=4, D3=10, P3=20). La tâche la plus prioritaire est la tâche T2 (D2 =5), et la tâche la moins prioritaire est la tâche T3 (D3 =10). La séquence est décrite sur l'intervalle
[0,20] = [0, ppcm(Pi)], et les trois tâches respectent leurs contraintes temporelles.[Dan04][MEH 14]

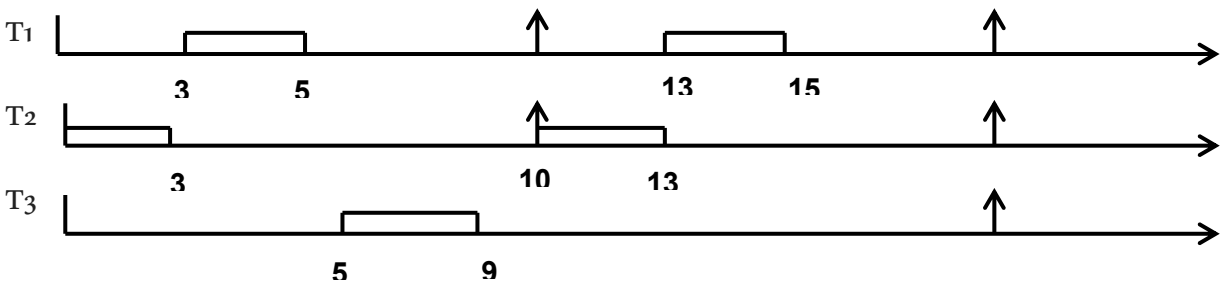

Figure 1.8 : "Inverse Deadline" (ID). [Dan04]

Remarque : L'algorithme Rate-monotonic, étant basé sur la notion d'échéance, cet algorithme s'applique aussi bien pour d'autres modèles de tâches que ceux des tâches à échéances sur

requêtes.

Avantages

- L'algorithme DM a les mêmes avantages de l'algorithme Rate Monotonic.
- Rate Monotonic pénalise les tâches peu fréquentes (longue période) mais urgentes (faible échéance). Deadline Monotonic s'avère meilleur dans ce cas.

Inconvénients

- L'algorithme DM a les mêmes Inconvénients de l'algorithme Rate Monotonic.[BOU 12]

II.8.1.3 "Least Laxity First" (LLF)

La laxité L(t) d'une tâche au moment t représente son retard maximum par rapport à son échéance pour (re) prendre son exécution, quand la tâche s'exécute seule : L(t) = D(t) -C(t). LLF est optimal dans la classe des algorithmes préemptifs pour des tâches périodiques indépendantes telles que Di ≤ Ti.

L'algorithme "least laxity first" attribue, à l'instant t, la plus haute priorité à la tâche ayant la plus petite laxité. Le calcul des laxités des tâches peut se faire :

• Aux dates de réveil des tâches : cas où la séquence produite est équivalente à celle produite par EDF,

• A chaque instant : cas qui entraîne plus de changements de contexte.

La figure 1.9 indique un ordonnancement donné par LLF des trois tâches périodiques : T1 (R1=0, C1=1, D1=10, P1=20), T2 (R2=0, C2=3, D2=9, P2=10) et T3 (R3=0, C3=5, D3=8, P3=10). A l'instant t =0 la tâche la plus prioritaire est la tâche T3 (L3=3), et la tâche la moins prioritaire est la tâche T1(L1=9).Le calcul des laxités est fait aux dates de réveil des tâches et, si deux tâches ont la même laxité au même instant, celle qui s'exécute en première est celle d'indice plus petit (comme le montre le cas de l'instant t =5 où les tâches T2 et T3 ont la même laxité, 2). La séquence est décrite sur l'intervalle [0,20] = [0, ppcm (Pi)], et les trois tâches respectent leurs contraintes temporelles.[Dan04] [MEH 14]

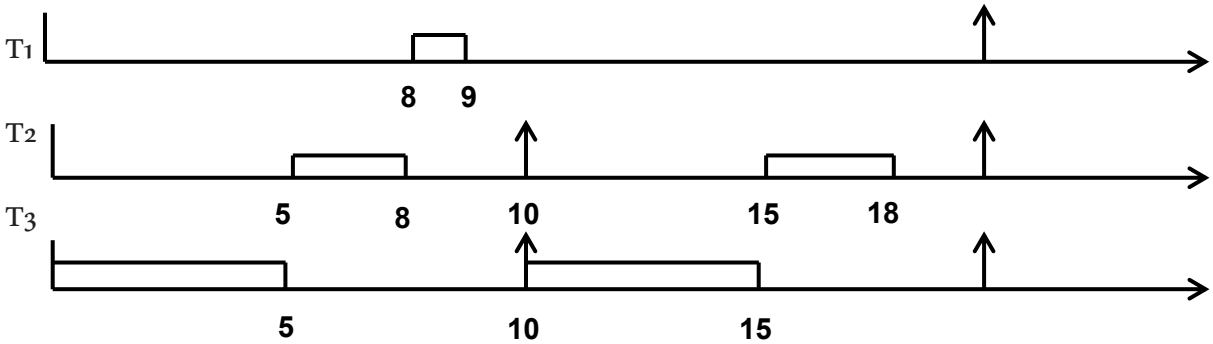

Figure 1.9: Least Laxity First (LLF) [Dan04]

Avantages

- Meilleur qu'EDF dans le cas de multi-processeur.

Inconvénients

- Forte complexité de la mise en œuvre : Difficulté du calcul du temps de calcul restant.
- Mauvais comportement en cas de surcharge.
- Nombre de préemptions engendrées supérieur. [BOU 12]

II.8.1.4 "Earliest Deadline First" (EDF)

C'est un algorithme qui est basé sur des priorités dynamiques (à toute instant, la priorité est inversement proportionnelle à la date de l'échéance) c'est-à-dire à chaque instant, la tâche la plus prioritaire est celle dont l'échéance absolue di est la plus proche. EDF est optimal dans la classe des algorithmes préemptifs pour des tâches périodiques indépendantes telles que Di ≤ Ti, et permet d'accepter un plus grand nombre de configurations de tâches que l'algorithme RM. [BOU 12]

La Figure 1.10 indique un ordonnancement donné par EDF des trois tâches périodiques : T1 (R1=0, C1=3, D1=7, P1=20), T2 (R2=0, C2=2, D2=4, P2=5) et T3 (R3=0, C3=1, D3=8, P3=10). A l'instant t =0 la tâche la plus prioritaire est la tâche T2 (D2=4), et la tâche la moins prioritaire est la tâche T3 (D3=8).

Les priorités des tâches, et donc l'ordre de leur exécution, évoluent les unes par rapport aux autres en fonction de leur urgence. La séquence est décrite sur l'intervalle [0,20] = [0, ppcm (Pi)] , et les trois tâches respectent leurs contraintes temporelles.[Dan04] [MEH 14]

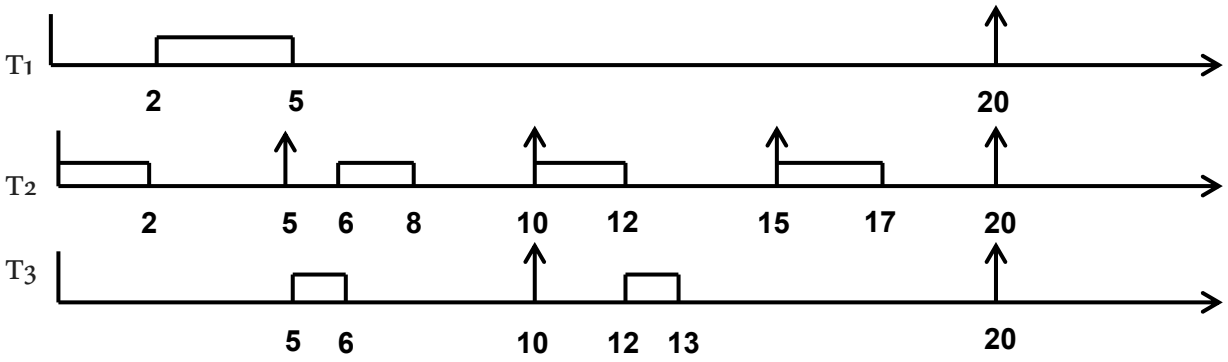

Figure 1.10: Earliest Deadline First (EDF) [DAN 04]

Avantages

- Utilisation possible de 100% du processeur.
- Meilleur que RM pour les tâches de courte échéance.
- Optimal pour les ordonnancements à priorité dynamique si les échéances sont inférieures

aux périodes.

Inconvénients

- Légère complexité de la mise en œuvre.
- Moins répandu dans les exécutifs que RM.
- Mauvais comportement en cas de surcharge. [BOU 12]

II.8.2 algorithmes d'ordonnancement sur une architecture multiprocesseur

II.8.2.1 "Rate-monotonic" (RM)

La Figure 1.11 indique l'ordonnancement « rate-monotonic » de quatre tâches périodiques à échéances sur requêtes : T1 (R1=0, C1=2, P1=5), T2 (R2=0, C2=1, P2=4) et T3 (R3=0, C3=2, P3=10), T4 (R4=0, C4=6, P4=20), avec 02 processeurs P1 et P2.
La tâche la plus prioritaire est la tâche T2 (P2 =5), et la tâche la moins prioritaire est la tâche T4 (P4=20). La séquence est décrite sur l'intervalle [0,20] = [0, ppcm (Pi)].[Dan04][MEH 14]

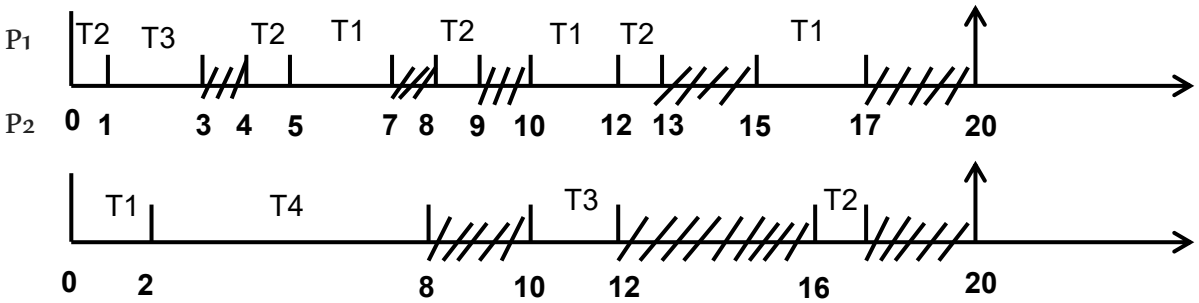

Figure 1.11 : l'algorithme « rate-monotonic » sur une architecture multiprocesseur [Dan04]

II.8.2.2 "Inverse Deadline" (ID) ou "Deadline Monotonic" (DM)

La Figure 1.12 indique l'ordonnancement « Inverse Deadline» de quatre tâches : T1 (R1=0, C1=2, D1=3, P1=5), T2 (R2=0, C2=1, D2=4, P2=4) et T3 (R3=0, C3=2, D3=9, P3=10), T4 (R4=0, C4=6, D4=18, P4=20), avec 02 processeurs P1 et P2.
La tâche la plus prioritaire est la tâche T1 (D1 =3), et la tâche la moins prioritaire est la tâche T4 (P4=18). La séquence est décrite sur l'intervalle [0,20] = [0, ppcm (Pi)].[Dan04][MEH 14]

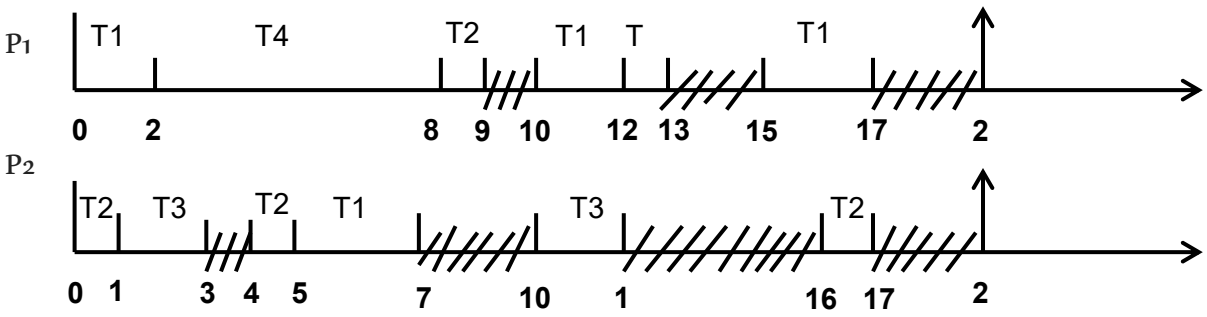

Figure 1.12 : l'algorithme «Inverse Deadlin (ID) » sur une architecture multiprocesseur [MEH 14]

II.8.2.3 "Least Laxity First" (LLF)

La Figure 1.13 indique l'ordonnancement « Least Laxity First» de quatre tâches : T1 (R1=0, C1=2, D1=3, P1=5), T2 (R2=0, C2=1, D2=9, P2=4) et T3 (R3=0, C3=2, D3=9, P3=10), T4(R4=0, C4=6, D4=18, P4=20), avec 02 processeurs P1 et P2.
La tâche la plus prioritaire est la tâche T1 (L1 =3), et la tâche la moins prioritaire est la tâche T4 (L4=12). La séquence est décrite sur l'intervalle [0,20] = [0, ppcm(Pi)].[Dan04][MEH 14]

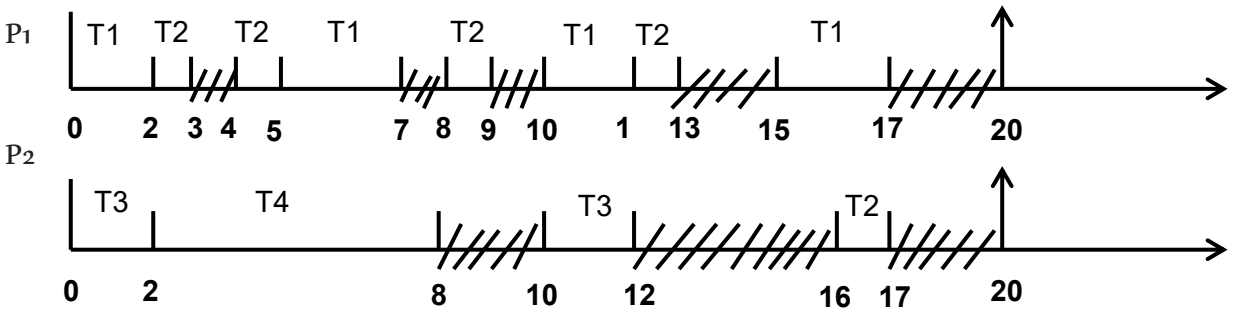

Figure 1.13: l'algorithme «Least Laxity First » sur une architecture multiprocesseur [MEH 14]

II.8.2.4 "Earliest Deadline First" (EDF)

La Figure 1.14 indique l'ordonnancement « Earliest Deadline First » de quatre tâches :
T1 (R1=0, C1=2, D1=3, P1=5), T2 (R2=0, C2=2, D2=9, P2=4) et T3 (R3=0, C3=1, D3=9, P3=10), T4 (R4=0, C4=10, D4=18, P4=20), avec 02 processeurs P1 et P2.
La tâche la plus prioritaire est la tâche T1 (D1 =3), et la tâche la moins prioritaire est la tâche T4 (D4=18). La séquence est décrite sur l'intervalle [0,20] = [0, ppcm(Pi)].[Dan04] [MEH 14]

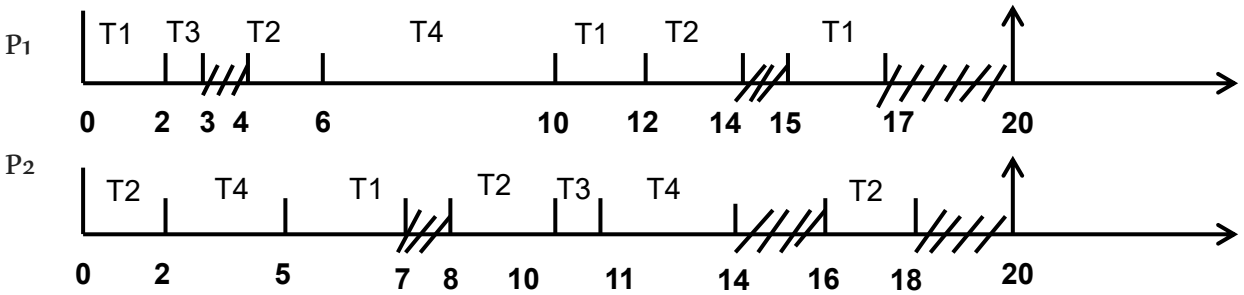

Figure 1.14 : l'algorithme «Earliest Deadline First » sur une architecture multiprocesseur [MEH 14]

Conclusion

Nous avons présenté dans ce chapitre, les notions. Les concepts de base ont été d'abord présentés, et ensuite les algorithmes principaux d'ordonnancement temps réel dans le cas d'une architecture monoprocesseur et multiprocesseur.

Chapitre 2

Estimation et réduction de la consommation d'énergie

Estimation et réduction de la consommation d'énergie

1. Introduction

La gestion de l'énergie est devenue un enjeu majeur dans nos sociétés modernes. La multiplication des équipements électroniques dans différents domaines (avionique, automobile, etc.) nécessite une prise en compte de cette problématique dans la conception des systèmes temps réel embarqués. [BOU 12]

La maîtrise de la consommation d'énergie est un problème largement abordé dans le domaine des systèmes embarqués. Avec le temps elle est devenue un facteur essentiel pour leur conception et leur complexité qui engendre une consommation énergétique de plus en plus élevée. La consommation d'énergie est un paramètre essentiel pour le développement des applications embarquées.

Dans ce chapitre nous nous intéressons à deux aspects, l'estimation d'énergie et la réduction de la consommation d'énergie au niveau du/des processeur(s), ainsi que des propositions technologiques pour minimiser cette consommation ont été présentées, en fin nous avons fait un état de l'art sur les travaux existant sur ce sujet. [MEH 14]

2. Présentation

Le développement informatique est en croissance infinie, et l'utilisation des appareils et des systèmes informatiques entrent dans tous les domaines de la vie, ça conduit à une augmentation croissante de l'énergie consommé par ces derniers, qui pose des préoccupations suivantes: L'utilisation répandue des appareils alimentés par batterie mobiles, ou le temps d'emploi dépend de la consommation de puissance du dispositif.

- La dissipation de puissance dans les grands systèmes informatiques mène à la haute interne températures qui a un impact négatif sur la fiabilité de ces derniers.
- Le coût élevé de l'énergie pour le fonctionnement et le refroidissement des centres de données volumineux.
- L'inquiétude générale au sujet des émissions de carbone du secteur des TIC, ce qui a la même grandeur que les émissions de gaz carbonique de l'industrie des transports aériens. Dans le passé, le nombre d'instructions exécutées par un système informatique dans une unité de temps (par exemple, MIPS ou FLOPS) était l'indicateur important pour mesurer la performance d'un système. Dans l'avenir, le nombre d'instructions exécutées par unité

d'énergie (par exemple, un joule) sera d'une importance égale. C'est-à-dire l'objectif de la conception des équipements électroniques n'est pas seulement de minimiser le temps de traitement, mais aussi de minimiser l'énergie consommé [KOP 13].

Parmi tous les composants électroniques, le processeur est particulièrement le plus utilisateur d'énergie, il pouvait à lui seul utiliser plus de 50 % de l'énergie lorsqu'il était sollicité intensivement. Donc pour réduire l'énergie du système, on joue sur la réduction de la fréquence de fonctionnement du processeur [MES 11].

3. Concepts de base

Le concept d'énergie est initialement introduit dans le domaine de la mécanique, se réfère à une quantité scalaire qui décrit la capacité de travail pouvant être effectué par un système. La puissance est l'intensité du travail « l'énergie » utilisée par unité de temps. L'énergie est donc l'intégrale de la puissance au cours du temps. Il existe de nombreuses unités différentes pour mesurer la quantité d'énergie. Nous allons utiliser le joule, qui est l'unité d'énergie dans le système MKS (mètre-kg-seconde). Un joule (J) est défini comme la quantité de travail effectué par une force d'un newton déplaçant une masse d'un kg sur une distance d'un mètre. Dans le même temps un joule est la quantité d'énergie qui est dissipée par la puissance d'un watt électrique pendant une seconde.

Une batterie est un dispositif de stockage d'énergie électrique. La tension qui existe entre les bornes de la batterie peut conduire un courant électrique à travers une résistance. Dans la résistance, l'énergie électrique est convertie en chaleur. Si un courant électrique I circule dans un fil avec une résistance R (en ohms), alors:$\boldsymbol{U = RI}$ **(1)**, selon la loi d'Ohm. La puissance électrique dissipée est donnée par : $\boldsymbol{W = UI}$ **(2)**. Où W représente la puissance électrique dissipée (en watts), I le courant (en ampères). Si une tension constante de U est appliqué à un système avec la résistance R sur une durée de temps t, l'énergie dissipée est égale à: $\boldsymbol{E = tU^2/R}$ **(3)**. Où E est l'énergie dissipée par l'effet de Joule dans, t qui désigne le temps en seconde set U et représente la tension et R la résistance, respectivement. [MEH 14][BOU 12]

4. Estimation de l'énergie

L'énergie qui est nécessaire par un dispositif informatique pour exécuter un programme peut être exprimée comme la somme des quatre types d'énergie suivantes :

$$\mathbf{E_{total} = E_{comp} + E_{comm} + E_{mem} + E_{IO}\ (4).}$$

Où E_{total} : est l'énergie totale nécessaire.

E_{comp} : désigne l'énergie nécessaire pour effectuer les calculs.

E_{mem} : dénote l'énergie consommée par le sous-système de mémoire.

E_{comm} : dénote l'énergie nécessaire à la communication.

E_{IO} : est l'énergie consommée par des dispositifs d'E / S [KOP 13].

Nous allons examiner les trois premiers termes dans le détail par la présentation de modèles énergétiques simplifiés. Les valeurs numériques des paramètres dépendent fortement de la technologie utilisée.

4.1 Energie de calcul

Énergie de calcul (ECOMP) se compose de deux parties :

a. L'énergie dynamique $\mathbf{E_{dynamique}}$ qui indique l'énergie nécessaire pour remplir les fonctions de commutation dans un circuit VLSI CMOS.

b. L'énergie statique $\mathbf{E_{static}}$ qui est dissipée grâce à un courant de fuite tirée de la source d'alimentation indépendamment de toute activité de commutation.

La consommation d'énergie dynamique d'une action de commutation d'un transistor peut être modélisée par un réseau de courant RC.

Dans un réseau RC, constitué par un bloc d'alimentation à la tension V (exprimée en volt), un condensateur C (Mesurée en farad), et une résistance R (mesurée en ohms). La tension sur la résistance R en fonction du temps t (en secondes) après l'événement de commutation est donnée par: $\boldsymbol{V_{res}(t) = Ve^{-t/\tau}}$ **(5).**

Où : τ = RC, qui a mesuré par seconde, est appelée la constante de temps. Cette constante de temps caractérisant la vitesse d'une opération de commutation.

L'énergie totale dissipée au cours d'une opération de commutation est donnée par l'intégrale de la tension produit V (t) et du courant I (t) à travers la résistance sur le période de temps à partir de zéro à l'infini.

$$\boldsymbol{E = \int_0^\infty V(t)I(t)\,dt = \frac{v^2}{R}\int_0^\infty e^{-2t/RC}\,dt = \frac{1}{2}CV^2} \quad \textbf{(6).}$$

Le courant qui circule dans le condensateur accumule une charge électrique dans le condensateur et ne contribue pas à la dissipation d'énergie.

L'énergie dynamique nécessaire à l'exécution d'une seule instruction couvre l'énergie nécessaire pour passer toutes les transitions de sortie d'une instruction et une énergie courte Eshort de circuit qui est évacuée par le courant bref débit de la source de tension, il peut être exprimée par l'équation suivant: $\boldsymbol{E_{Dynamic-Instruction} = C_{eff}V^2}$ **(7).**

Tel qu'est $\boldsymbol{C_{eff}}$ la capacité effective hypothétique : $\boldsymbol{C_{eff} = C/2 + E_{short}/V^2}$ **(8).**

L'énergie dynamique nécessaire pour exécuter un programme avec N instructions est donnée par: $\boldsymbol{E_{Dynamic-Programme} = C_{eff}V^2N}$ **(9).**

Le terme IP-core (core de la propriété intellectuelle) est introduit pour désigner une unité fonctionnelle bien déterminée d'un système sur puce (SoC). Si l'IP-core s'exécute sur au moyen une instruction par cycle d'horloge, et ce noyau IP est entraîné avec une fréquence f.la dissipation de puissance dynamique de l'IP-core égal **:** $\boldsymbol{P_{dynamic} = C_{eff}V^2f}$ **(10).** [KOP 13][BOU12]

4.2 Energie de communication

L'énergie de communication est l'énergie requise pour la transmission d'un flux de bits à partir d'un expéditeur à un destinataire. Cette énergie est fortement asymétrique car l'expéditeur doit générer des signaux qui sont assez forts pour atteindre le destinataire prévu, alors que le récepteur doit seulement détecter les signaux entrants (faible).

L'énergie nécessaire par un expéditeur $\boldsymbol{E_{TX}}$ pour transporter une chaîne de bits de k octets peut être approximée par : $\boldsymbol{E_{tx}(k) = E_o + K(Ec + 8E_{tr})}$ **(11)** ;

Tandis que l'énergie l'Erx nécessaire à récepteur peut être approchée par : $\boldsymbol{E_{rx}(k) = E_o + K(Ec)}$ **(12).**

Où **Eo** : est l'énergie nécessaire pour mettre en place l'envoi ou la réception d'un message.

Ec : est l'énergie dissipée par le dispositif de contrôle de communication.

E_{tr} : dénote l'énergie nécessaire pour transmettre un bit unique. Il est différent pour la communication filaire et sans fil unique. [KOP 13]

4.3 Energie de mémoire

Dans les applications gourmandes en mémoire, l'énergie qui est nécessaire pour accéder eux sous-systèmes de mémoire peut être supérieure à l'énergie qui est nécessaire pour les calculs L'énergie qui est consommée par un sous-système de mémoire se compose de deux parts :

a. Le premier part est le produit de la puissance qui est consommée par le sous-système de mémoire, et le moment où le sous-système de mémoire est sous tension. Ce terme dépend du type et

de la taille de la mémoire.

b. Le deuxième part est constitué du produit des nombres d'accès mémoire et de l'exigence d'énergie pour une seule mémoire accéder. [KOP 13]

5. Techniques de réduction de la consommation

La consommation d'énergie par les appareils informatiques fait partie des plus importants des effets secondaires. D'un point de vue algorithmique, ceci inclut l'étude de problèmes d'ordonnancement comme un objectif pour la minimisation de l'énergie consommée par les processeurs. Cette consommation d'énergie est causée principalement par l'exécution des différentes tâches qui sont soumises aux machines. Cependant, une partie non négligeable de la dépense énergétique provient aussi de l'inactivité des machines puisque même en l'absence de programmes ou requêtes à exécuter, une machine dépense une quantité constante d'énergie que nous appelons énergie statique (ou énergie de base).

- Une première stratégie est de travailler sur la technologie des composants matériels. Ainsi une diminution de la taille des composants, rendue possible par des progrès dans les techniques de fabrication, permet une tension d'alimentation plus faible et donc une consommation moindre.
- Une seconde stratégie est de limiter l'alimentation d'un composant aux blocs nécessaires le traitement en cours, par exemple, on peut diviser une mémoire cache en des blocs pouvant être activés indépendamment les uns des autres.

- Une autre possibilité est de limiter le nombre de changements d'états dans un circuit car chaque changement d'état induit un coût énergétique.

- Une autre voie qui est explorée est de spécialiser les composants pour l'usage fait, par exemple à l'aide de composants reconfigurables FPGA.

- Certains travaux étudient l'utilisation de circuits électroniques asynchrones qui contrairement aux circuits synchrones traditionnels, présentent la caractéristique intéressante de ne consommer de l'énergie que dans les sous-parties du circuit réellement utilisées lors de l'exécution d'une instruction. [MEH 14]

6. Contraintes de temps et d'énergie

L'apparition de composants électroniques à tension d'alimentation variable constitue un progrès important dans l'industrie des processeurs, à l'heure actuelle, de nombreux processeurs comportant cette possibilité sont disponibles commercialement.

La puissance dynamique dissipée par un composant variant au minimum en le cube de la fréquence de fonctionnement, donc il est judicieux de fonctionner le processeur à la fréquence la plus faible compatible avec le niveau de performance requis. Mais, lorsque des contraintes de temps explicites pèsent sur certaines activités du système, il s'agit naturellement de les respecter avec l'objectif supplémentaire de minimiser la consommation en énergie.

Donc le problème d'ordonnancement à résoudre consiste non seulement à déterminer l'ordre dans lequel exécuter les activités du système, mais également à fixer la fréquence de fonctionnement du processeur au cours du temps.

Si aucune contrainte de temps n'est spécifiée alors on met le processeur en veille ce qui naturellement est incompatible avec le niveau de performances minimales attendu. Une technique possible pour garantir le bon fonctionnement du système est alors d'allouer à chacune des activités une date d'échéance. Une autre possibilité pour obtenir les performances minimales est de ne pas considérer des échéances individuelles, mais de raisonner en termes de nombre minimal de requêtes traitées par unité de temps, afin de maximiser le nombre de requêtes traités [BOU 12].

7. La technique DVS (Dynamic Voltage Scaling) (Adaptation de voltage)

Le moyen le plus efficace pour abaisser la consommation d'énergie est connu sous le nom de Dynamic Voltage Scaling (DVS). De nombreux processeurs modernes peuvent dynamiquement abaisser la tension pour réduire la consommation d'énergie.

Le retard de propagation limite la fréquence d'horloge du microprocesseur: le processeur peut fonctionner à une tension d'alimentation plus faible, seulement si la fréquence d'horloge est réduite à supporter l'augmentation de retard de propagation. Dans la plupart des cas, lorsque la réduction de la tension d'alimentation est nécessaire pour abaisser la fréquence de fonctionnement (par exemple, la vitesse du microprocesseur) alors toutes les tâches prendront plus de temps à être exécuté. Dans les systèmes temps réel, si le changement de fréquence n'est pas fait correctement, les exigences de synchronisation de l'application peuvent ne pas être respectées. Par conséquent, les avantages de la technique DVS peuvent être exploités dans les systèmes temps-réel seulement après une identification minutieuse des conditions dans lesquelles nous pouvons ralentir en toute sécurité le processeur sans manquer aucune échéance (pour les tâches temps réel dur) ou manquant d'un nombre limité des échéances.

Les algorithmes d'ordonnancement prenant en compte la réduction de la consommation peuvent exploiter la technique DVS en sélectionnant, en plus de la tâche à programmer, la

fréquence de fonctionnement du processeur à chaque instant. Le problème devient plus difficile dans les systèmes avec une combinaison de tâches dures et douces, périodiques et apériodiques en temps réel.

Les systèmes temps réel à vitesse variable sont toujours décrits à l'aide de graphiques 'Gantt' comme le montre la figure 2.1. Lors de la conception d'un système temps réel, la première préoccupation est généralement le temps, laissant l'efficacité énergétique comme une conséquence espoir de décisions empiriques.

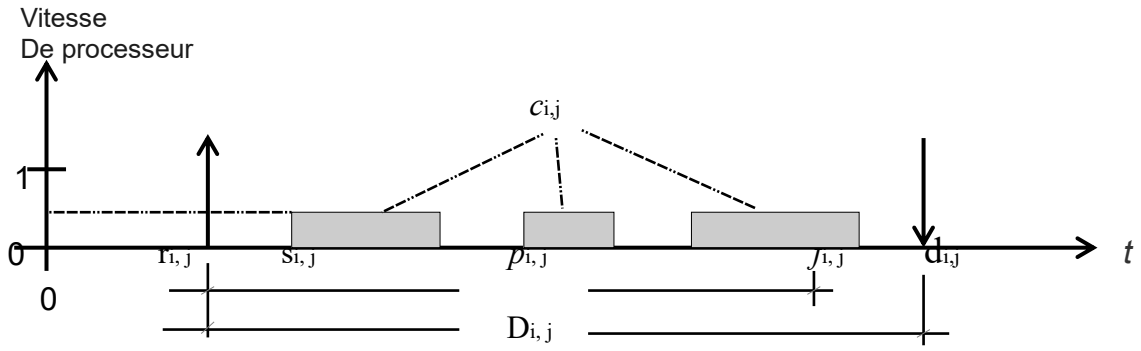

Figure2.1: Un diagramme de GANNT décrivant le travail générique τi, k. [MEH 14]

8. Une taxonomie des algorithmes d'ordonnancement prenant en compte la réduction de la consommation

Récemment, de nombreux algorithmes ont été proposés dans la littérature pour exploiter les processeurs de tension variable. Dans ces algorithmes, l'ordonnancement sélectionne la tâche d'exécution, en plus il permet également de sélectionner la fréquence du microprocesseur. Le problème devient plus difficile dans les systèmes avec une combinaison des tâches en temps dures et souple, périodiques et apériodiques. Pour cette raison, la plupart des algorithmes se concentrent seulement sur un seul type de tâche (en général, des tâches temps réel périodiques dures), en évitant le cas le plus difficile de la coexistence de différents types de tâches dans le même système. Cependant en pratique, de nombreux systèmes réels sont constitués d'un mélange de tâches dures et souples. Nous présentons maintenant une taxonomie possible d'algorithmes d'ordonnancement. [KIM 02].

Les algorithmes peuvent être divisés en deux techniques : statiques et dynamiques, en fonction des décisions de mise à l'échelle il sera mis hors ligne ou en ligne, respectivement. Noter que statique est synonyme de hors-ligne, alors que dynamique est synonyme de en ligne. En effet, dans le premier cas, le système d'exploitation choisit statiquement la vitesse du

processeur hors ligne, quel que soit le comportement des tâches à 'exécuter. Dans le dernier cas, le système choisit la vitesse du processeur de manière dynamique au moment de l'exécution. Il s'agit de la notation typique utilisée dans la littérature en temps réel. [MEH 14]

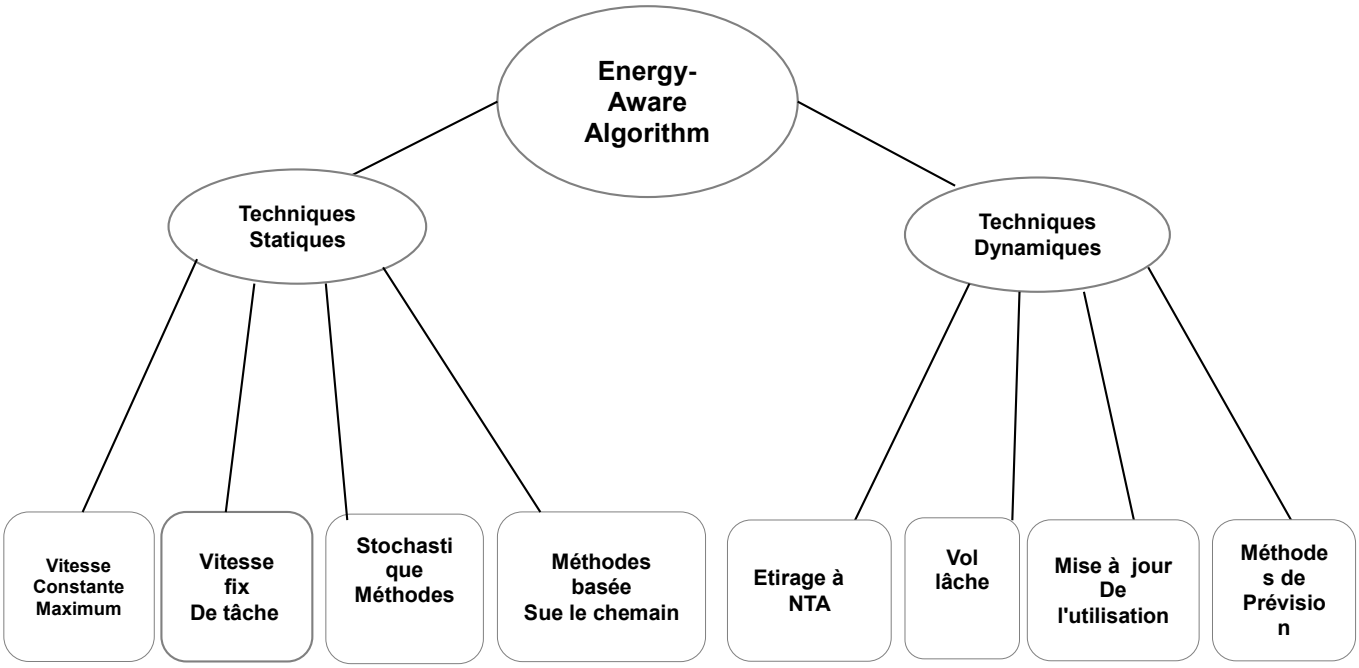

Figure 2.2: Taxonomie des algorithmes d'ordonnancement energy-aware [MEH 14]

8.1 Les techniques statiques

Les techniques statiques peuvent être divisées en quatre catégories, selon le moment où la vitessedu processeur est déterminée. Ces classes sont les suivantes: vitesse maximale constante, la vitesse de travail fixe, les méthodes stochastiques et les méthodes basées sur le chemin.[MEH 14]

8.1.1 Vitesse constante maximale

Dans cette première classe des méthodes, une seule vitesse constante est calculée hors ligne et affectée pour exécuter toutes les tâches jusqu'à ce que le jeu de tâche change. Cette vitesse est définie comme étant la fréquence la plus basse possible qui garantit le calendrier possible d'un ensemble de tâches. Cette technique est aussi appelée «Affectation de la tension du système fixe". Avec cette approche, il n'y a aucune charge supplémentaire pour la commutation de tension au moment de l'exécution.

Toutefois, étant donné les contraintes de temps de toutes les tâches qui doivent être satisfaisantes, la fréquence de base doit être la plus haute fréquence au pire de cas requise par le jeu de tâche, qui peut avoir comme conséquence moins d'économiser l'énergie comparée à d'autres approches. [MEH 14]

8.1.2 Vitesse de tâche fixe

Cette deuxième classe de méthodes statiques parfois s'appelle « l'affectation fixe de tension de tâche » et appartient à la classe des techniques de mise à l'échelle inter-tâches, ce qui signifie que la vitesse du processeur n'est pas fixe mais statiquement assignée sur une base de tâche-par-tâche (c.-à-d., basé sur les paramètres de la tâche) avant l'exécution. En d'autres termes, étant donné un ensemble de tâches périodiques, l'algorithme assigne une fréquence qui peut être différente pour chaque tâche individuelle. Les affectations sont toujours calculées hors ligne, et sont fixés jusqu'à ce que le jeu de tâche change. Puisque l'ordonnancement de tâche est périodique, l'ordonnanceur de tension obtenu par cette méthode est également périodique et peut être enregistré dans une table. Cette technique absorbe un temps système supplémentaire pour mesurer la fréquence de processeur aux valeurs correctes pendant un commutateur de contexte. [SAO 03]

8.1.3 Méthode stochastique

Cette méthode appartient à la classe des techniques de mise à l'échelle intra-tâche ce qui signifie que la vitesse de traitement est réglée dans une limite de tâche individuelle (par exemple, dans l'exécution d'une tâche unique).Cette méthode est basée sur l'idée qu'il vaut mieux de reporter un certain travail dans l'espoir que la tache actuelle aura un temps d'exécution moins que la condition au pire de cas. Ainsi, si la tâche se termine plus tôt que son WCET (Worst Case Execution Time), la haute vitesse peut ne jamais être nécessaire. La méthode commence l'exécution de tâche à une vitesse réduite, et accélère la vitesse du processeur pendant l'exécution de tâche. La vitesse d'horloge est soulevée au cas de temps spécifique, jusqu'à ce que le travail se termine. Théoriquement, si la fonction de densité de probabilité du temps d'exécution de tâche est connue à l'avance, l'ordonnancement de vitesse optimal peut être calculé [MEH 14].

8.1.4 Méthodes basées sur le chemin

Ces méthodes font partie des techniques de mise à l'échelle intra-tâche. Dans les méthodes, basées sur le chemin, la fréquence du processeur est basée sur un chemin d'exécution de référence prévu, tel que le chemin d'exécution au pire de cas ('WCEP' Worst Case Execution Path). Considérons les deux branchements d'un type if-then-else, qui peuvent être très différentes des temps de calcul. Si l'application dévie du plus long chemin d'exécution, alors la fréquence d'opération peut être abaissée pour réduire la consommation d'énergie. Les emplacements pour le passage de vitesse possible à l'intérieur du programme peuvent être identifiés par analyse statique de programmes ou de l'exécution du profilage [MEH 14].

8.2 Techniques dynamiques

Les algorithmes dynamiques peuvent récupérer le temps libre dynamique, ayant pour

résultat une
quantité plus grande d'énergie économisée. Ce temps supplémentaire peut être exploité par l'algorithme dynamique pour modifier le calendrier au moment de l'exécution et réduire plus la vitesse du processeur. Les méthodes dynamiques peuvent exploiter des informations sur le comportement d'exécution des tâches, qui peuvent être très loin des suppositions pessimistes exigées au temps de modèle. Pour cette raison, elles laissent théoriquement réduire la consommation d'énergie par une plus longue valeur que des arrangements statiques.Les algorithmes dynamiques sont typiquement basés sur la récupération de temps lâche. La plupart des algorithmes utilisent au moins une des techniques d'estimation de temps lâches suivantes.

8.2.1 Étirage à NTA

Cette méthode simple d'estimation du temps libre dynamique, consiste à calculer le suivant temps d'arrivé (NTA), qui est le temps de sortie de la tâche suivante. Le travail $\tau_{i,j}$ est prévue pour l'exécution. Si à ce moment, le NTA est plus tard (à + WCETi), alors l'exécution d' τi, j peut être "étiré" jusqu'à NTA (i.e., la vitesse du processeur peut être réduite de sorte que l'exécution de τi, j dans le pire des cas se termine exactement au NTA). Cela permet de réduire la fréquence du processeur dans l'intervalle de temps [pour le NTA], comme le montre la figure 2.3.

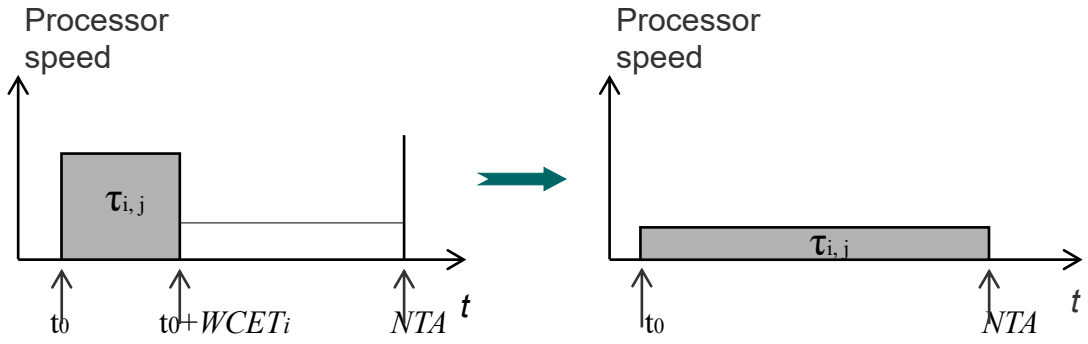

Figure 2.3 :L'étirement à la technique NTA [MEH 14].

8.2.2 Vol lâche

L'idée de base de cette méthode est quand une tâche termine son exécution plus tôt que son exigence pire de cas, les tâches suivantes peuvent utiliser le temps libre. Typiquement, les tâches récupèrent seulement le temps libre des tâches de haute priorité pour éviter des algorithmes de calcul coûteux.[MEH 14]

8.2.3 Mise à jour d'utilisation

Cette technique estime l'utilisation requise actuelle à n'importe quel instant du temps.

Chaque
fois qu'une telle utilisation change, la fréquence de processeur est changée en conséquence. Le principal avantage de cette méthode est la mise en œuvre simple. En fait, il ne fait aucune hypothèse sur la durée de tâche, et attend l'accomplissement de tâche pour savoir le temps d'exécution exacte. Puis, il change la vitesse du processeur selon cette information. [SOA 03]

8.2.4 Méthodes de prévision

Si l'ordonnanceur temps réel savait à l'avance qu'une tâche compléterait l'exécution avant son
temps d'exécution de pire cas, elle exploiterait cette information pour réduire davantage la consommation d'énergie. Ainsi, cette classe des techniques prend des décisions à l'avance, avant que les cas actuels des tâches aient complétés. Typiquement, ces algorithmes essayent de reporter un certain travail dans l'espoir que les cas actuels exécuteront moins que le scénario au pire de cas. [MEH 14][SAE 03]

9. Etat de l'art

Généralement, la minimisation de la consommation d'énergie est en opposition avec les critères
d'optimisation classiques qui traduisent la qualité d'un ordonnancement donné. Dans ce contexte, une multitude de problèmes ont été étudiés.

[RAM 17], Bien que la consommation due à l'exécution d'un programme soit difficile à estimer, il existe une relation entre le nombre d'instructions exécutées et la consommation D'une manière générale, plus le nombre d'instructions est élevé, plus la consommation est importante, les techniques d'optimisation logicielles peuvent donc aider à diminuer cette consommation. La majorité des développements se font actuellement avec des langages dits de « haut-niveau », comme le langage C ou Ada. Les développements en langage de « bas-niveau » comme l'assembleur étant limités aux applications ayant des contraintes extrêmement fortes sur leurs performances et leurs fiabilités (programmation de DSP ou de codes critiques). L'avantage des langages de « haut-niveau » est de permettre au programmeur d'écrire son application dans un langage facilement compréhensible et de plus portable, la génération de code machine pour une architecture donnée étant réalisée de manière automatique par un compilateur. Cette méthode permet d'améliorer la vitesse de développement, mais cela s'effectue au dépend de la qualité des codes machines générés.Il est toujours possible d'optimiser « à la main » des codes ainsi générés, mais de tels codes ne

sont généralement pas très lisibles leur manipulation est donc fastidieuse, et la taille importante des applications actuelles rend leur analyse encore plus complexe. Cependant, sur de petits programmes, il peut être envisagé d'avoir recours à une telle méthode**,** où les opérations de mémoire à mémoire sont remplacées par des opérations de registre à registre, moins coûteuses en énergie. Cette optimisation permet de diminuer de 403 la consommation totale du programme, mais il s'agit d'un cas précis, et il n'est pas justifié de prendre cette valeur comme référence. Les optimisations de code manuelles étant difficile à mettre en œuvre, les techniques décrites ci-dessous portent sur des optimisations automatiques au niveau du générateur de code. Elles n'atteignent pas le même niveau de performances mais elles sont nettement plus simples et peuvent être aisément implantées dans une chaîne de développement.

Dans **[ACD 17]**, ENT est un langage de programmation, qui permet de mettre en place une gestion de l'énergie au niveau de l'application, de manière proactive et adaptative. La conception proactive permet aux programmeurs, d'appliquer leurs connaissances de l'application à la gestion de l'énergie, en caractérisant statiquement le comportement énergétique de différentes parties du code dans différents modes de fonctionnement. La conception adaptative permet de retarder cette caractérisation jusqu'à l'exécution, ce qui permet de capturer dynamiquement le comportement du programme en fonction de son état, de ses paramètres de configurations, le niveau de la batterie ou la température CPU.

Dans **[ZLX 15]**, L'efficacité énergétique d'un système peut être augmentée en installant des éléments mémoires pilotés par logiciel à l'intérieur des microprocesseurs (on chip memory). Cette architecture est appelée Scratch Pad Memory (SPM). Une autre mémoire « on chip » existe, appelée mémoire cache. Cependant, comparée à la mémoire cache, la SPM utilise moins d'énergie pour accéder à l'information. Les microprocesseurs utilisent les SPM via différentes techniques. L'un des principes étant d'analyser le code et de stocker l'information là où l'efficacité énergétique sera la meilleure. Des études actuellement menées sur le sujet proposent des méthodes combinant l'utilisation des SPM avec la mémoire cache.

Dans **[QJY 14]**, Par rapport aux batteries, les condensateurs présentent le gros avantage d'avoir une durée de vie bien supérieure, en termes de nombre de cycles de rechargements. Bien que les travaux de Qianao Ju et al. Aient aussi pointé des inconvénients sur les supercondensateurs, il propose un algorithme pour en diminuer les effets. Ces composants sont utilisés de différentes

manières. Une première méthode est l'utilisation d'un ou plusieurs supercondensateurs pour stocker une charge qui sera directement utilisée selon les besoins du système. Si besoin en utilisant des algorithmes de gestion d'énergie. Ensuite, d'autres condensateurs peuvent être utilisés pour alimenter individuellement des périphériques ou un système de régulation d'énergie.

Dans **[Ann 13]**, propose un algorithme d'ordonnancement PbS. L'algorithme proposé de PbS,
et sa variation, appelée l'algorithme d'E-PbS, sont basées sur l'algorithme d'ordonnancement dirigé par la force utilisé dans la synthèse de haut niveau des circuits digitaux. Les algorithmes proposés sont comparés aux approches d'ordonnancement classique comme FCFS (SAS) et EDF (SbS). Les algorithmes de SAS et de SbS mettent également en application une cartographie explicite, ou des algorithmes de graduation de tension, après et avant l'ordonnancement du programme et compare aux algorithmes d'ordonnancements classiques. Les algorithmes de PbS et d'E-PbS fournissent une bonne utilisation de processeur et une consommation d'énergie inférieure, comparées aux algorithmes de SAS et de SbS.

Dans **[ALB 11]**, Albers et al ont proposé l'ordonnancement des tâches avec des dates d'arrivées et des dates d'échéances quelconques sur un ensemble de machines identiques avec migration et dans le but de minimiser l'énergie totale consommée. Albers et al. Ont utilisé une approche combinatoire basée sur le calcul de flots maximums et ils ont proposé un algorithme polynomial.

Dans **[ALB 07]**, Albers et Fujiwara se sont intéressés au problème de minimisation du temps de réponse total plus l'énergie consommée. En utilisant une approche de programmation dynamique, les auteurs ont décrit un algorithme optimal dans le cas horsligne. Dans le cas on-line ils ont présenté une solution compétitive si les quantités de travail sont unitaires. Ils ont également montré qu'aucun algorithme on-line ne peut réaliser un rapport de compétitivité réaliser un rapport de compétitivité constant si les quantités de travail sont arbitraires.

Dans **[MNW 01]**, Les appels de procédures sont couteux dans la plupart des architectures. La technique d'inlining des procédures peut aider à améliorer les performances et à économiser l'énergie logicielle en diminuant les surcharges (overhead) associés aux appels et aux retours. Un effet secondaire de l'inlining est l'augmentation de la taille du code. Cette augmentation nécessite en conséquence une taille mémoire plus importante, qui conduira à une plus grande dissipation

d'énergie.

Conclusion

Dans ce chapitre nous avons présenté les concepts de base relatifs à l'estimation de la consommation d'énergie et un tour d'horizon des techniques d'estimation et de réduction d'énergie consommée. Ensuite nous avons traité le moyen le plus efficace pour abaisser la consommation d'énergie qui est connu sous le nom de Dynamic Voltage Scaling (DVS). Une taxonomie des algorithmes d'ordonnancement prenant en compte la consommation a été consommation a été évoquée les travaux. En fin, un état de l'art et synthèse sur les travaux existant, et particulièrement orientés vers les tâches périodiques indépendantes, a été mis en question.

Chapitre 3

Contribution

Contribution

1. Introduction

Dans ce chapitre, nous essayons de présenter notre nouvelle approche qui consiste à formuler le problème d'optimisation de la consommation d'énergie sous contraintes temporelles à l'aide de la technique (DVS) et l'algorithme d'ordonnancement EDF*.

Notre problème se compose des tâches périodiques indépendantes et appliquée sur une architecture multiprocesseurs. Toutes les étapes de la solution seront discutées ainsi que l'implémentation avec un exemple explicatif.

2. Définition du problème

Les systèmes embarqués composent une grande partie des appareils utilisés quotidiennement. Ils sont des systèmes informatiques électroniques dirigés vers une tâche précise, et qui fonctionnent en temps réel.

Lors de la conception des systèmes embarqués, il faut prendre en considération deux problèmes contradictoires : la minimisation de temps d'exécution et l'énergie consommée : Il faut concevoir un système embarqué de telle sorte que les tâches ne dépassent pas leurs échéances tout en minimisant la consommation énergétique.

Le problème d'ordonnancement dans un système embarqué temps réel est un problème qui consiste à allouer *n* tâches de manière ordonnée sur *m* processeurs de telle sorte que les contraintes temporelles soient respectées tout en minimisant l'énergie consommée.

Le but sera ici de montrer comment modéliser le problème à l'aide de l'approche proposé.

3. Consommation optimale de l'énergie dans un modèle de tâches périodiques indépendantes :

3.1 Notations, définitions, concepts :

Une tâche correspond à une application utilisateur du système et chaque tâche est développée par l'utilisateur du système pour répondre à un besoin. Une tâche T est définie comme l'exécution d'une suite d'instructions. Nous supposons que toutes les tâches sont indépendantes, périodique que l'ordre dans lequel les tâches sont exécutées n'a pas de conséquence sur la bonne exécution du système du moment qu'elles respectent leurs contraintes temporelles.[LEG 14]
Chaque tâche a des caractéristiques spécifiques telle que :

i : l'identité de la tâche

Ri : la date d'arrivée de la tâche Ti

Pi : la période de la tâche Ti

ti : le temps d'exécution de la tâche Ti.

Ci : le coût pire cas d'exécution de la tâche Ti en cycles processeur

Di : l'échéance de la tâche Ti (Deadline)

gi(S) : la puissance consommée pour l'exécution de la tâche Ti.

Smin : la vitesse minimale qui assure le fonctionnement de système.

Smax : la vitesse maximale du processeur (normalisée=1).

NRsti(t) : le temps d'exécution restant pour la tâche Ti au moment t avec la vitesse nominale S.

PRsti(t) : le temps d'exécution au pire de cas restant pour la tâche Ti au moment t avec la vitesse d'exécution Si.

Le temps d'exécution est donné par : $t_i = C_i/S_i$ *(1)*

-On considère que le changement de la vitesse du processeur peut se produire durant toute la période P;

-La condition d'exécution de chaque tâche est : $C_i/S_{max} \leq (D_i-R_i)$ *(2)*

Pour chaque itération j de la tâche Ti : $C_{ij}/S_{max} \leq (D_i-R_i)$ *(3)*

On résout le problème pour une période P tel que **P= PPCM (P1, P2….. Pn)** *(4)* c'est à dire la période d'étude = **[0, PPCM (Pi)]**.

L'idée d'une solution dynamique, repose sur le principe d'ajuster la vitesse des tâches à venir en fonction du gain de temps qui se produit généralement par la terminaison en avance par rapport a l'estimation au pire de cas, tout en respectant leurs échéances estimées.

-Si la vitesse d'exécution Si, Si=Smin ou Si = S', alors la consommation d'énergie est optimale sachant que S'est la vitesse d'exécution qui assure une charge du processeur égale 1, c.-à-dire la vitesse qui vérifie l'égalité suivant :$\sum_{t=1}^{n} \frac{ti}{pi} = 1$ tel que **$t_i = C_i/S_i$** *(5).*

Alors la vitesse nominale **S"=max {Smin, S'}** *(6)*.

-L'ordre des tâches dans la file d'attente FA est calculée par l'algorithme EDF*. [MEH 14]

-L'utilisation d'une tâche est le rapport entre son WCET et sa période.

-L'utilisation Ui de la tâche Ti est donc $\boldsymbol{Ui = \frac{Ci}{Ti}}$ *(7).*

-L'utilisation globale U de l'ensemble de tâches T est la somme de toutes les utilisations individuelles des tâches de l'ensemble de tâches**:** $\boldsymbol{U = \sum_{i=1}^{n} \frac{Ci}{Ti}}$ *(8).*

3.2 Approche

L'idée de la solution est d'exécuter *n* taches sur *m* processeurs tels que :

- l'exécution des tâches sur une plate-forme multiprocesseur à mémoire commune.
- Le fonctionnement des processeurs est basé sur une horloge commune.
- la vitesse d'exécution d'une tâche pourrait donc changer à chaque nouvelle attribution du processeur, c'est-à-dire au retour de chaque préemption.

Pour les tâches dans la file d'attente, nous adoptons la politique d'ordonnancement EDF*, à savoir la tâche qui a la plus courte distance limite sera fixée en premier. Cet algorithme assigne des priorités en fonction de la proximité temporelle de l'échéance de chaque tâche, ainsi la tâche ayant l'échéance la plus proche se voit attribuée la plus haute priorité. A tout instant, c'est la tâche ayant la plus haute priorité qui s'exécute .La priorité de chaque tâche est recalculée dynamiquement à chaque modification de l'état du système (arrivée d'une tâche, terminaison d'une tâche).

Et pour la vitesse, nous calculons la vitesse d'exécution de chaque tache entrée par rapport a la date d'arrivée et on prend en considération l'intervalle de la vitesse V= [0.3, 1].

N'oubliant pas que lors de l'exécution parallèle de taches, l'ordonnanceur doit affecter les taches aux processeurs de manière équitable, le processeur que n'est pas en train d'exécuter des taches n'est pas forcement qu'il est intéressé par une affectation immédiate de tache, l'algorithme d'ordonnancement doit gérer l'équité d'affectation des taches aux processeurs en parallèles.

3.3 La politique d'ordonnancement :

L'ordonnanceur c'est lui qui gère la politique d'exécution des tâches dans notre cas il choisit d'abord les tâches ayant un deadline courte (algorithme EDF), puis à chaque date d'horloge il calcule les vitesse d'exécution des tâches pour toutes les tâches en attente, pour adapter la vitesse de processeur à cette vitesse dont le but est de diminuer la fréquence de processeur c'est à dire diminuer l'énergie de son consommation, si une taches ayant une date deadline prioritaires et il existe un processeur chômé , l'ordonnanceur va exécuté cette tâche, cette politique applicable pour plusieurs tâches en parallèle selon le nombre de processeur disponibles.

3.4 Adaptation dynamique de la vitesse

La réalisation d'une adaptation dynamique de la tension nécessite d'une part un matériel capable d'ajuster sa fréquence et sa tension à la demande et d'autre part un algorithme capable de déterminer quel doit être la fréquence de fonctionnement courante.

Dans un premier temps, les contraintes matérielles pour la mise en œuvre d'un tel

mécanisme seront abordées. Ensuite plusieurs algorithmes d'adaptation de fréquence sont étudies, certains ne prenant en compte aucune contrainte temporelle, et d'autres cherchant à adapter la fréquence tout en respectant les contraintes temps réel que peuvent avoir certaines applications.

Dans [WWD 94], l'algorithme basé sur l'EDF incluant la variation de fréquence du processeur dans l'ordonnancement de tâches temps-réel. Cet algorithme se base sur l'ordonnancement EDF et gère des tâches périodiques. Le test d'acceptation d'une nouvelle tâche dans le système est en fait un test de faisabilité de l'ordonnancement EDF, il vérifie donc qu'à sa fréquence maximale, le processeur sera capable d'exécuter toutes les tâches. L'ordonnancement des tâches est effectué en fonction de leurs échéances, mais un calcul supplémentaire est effectué pour fixer la fréquence de fonctionnement du processeur. Pour cela, l'ordonnanceur calcule, pour chaque tâche j, le paramètre Uj qui correspond au plus petit taux d'utilisation du processeur permettant de respecter l'échéance de cette tâche. Ce taux dépend de l'échéance de cette tâche, de son temps d'exécution ainsi que du temps d'exécution de toutes les tâches précédant cette tâche. Ensuite, lors de l'exécution des tâches, l'ordonnanceur fixe la fréquence de fonctionnement du processeur en fonction de la plus grande des valeurs des Uj des tâches qui restent à exécuter (la fréquence peut donc être modifiée après chaque terminaison de tâche).

3.5 Objectifs

Les objectifs de l'algorithme utilisé pour la solution sont :

1- produire un ordonnancement qui, dans la mesure du possible, permet à toutes les tâches de Respecter leur échéance;

2- produire un ordonnancement qui optimise la consommation globale de la plate-forme.

3.6 Principe

Le paramètre **NBR_PROC** désigne le nombre de processeurs présents sur la plate-forme matérielle.

L'ensemble des tâches est regroupé dans une liste nommée ‘ **FAT‘**(file d'attente des tâches) et classée par l'algorithme EDF*.

Avant l'insertion d'une tâche Ti on doit simuler l'insertion de cette tâche dans **‘FAT’** et on doit aussi évalue ce placement dans liste, c'est-à-dire La possibilité éventuelle avec lequel la tâche Ti termine son exécution et respecte son échéance (dans le cas d'utilisation de la vitesse maximale Vmax, avec tous les tâches). La tâche Ti peut être placée devant des tâches déjà affectées.

Le nouveau taux d'utilisation des processeurs est augmenté par l'insertion de cette tâche et la

manière dont le placement de cette tâche.

On prend en considération des contraintes de l'algorithme d'ordonnancement :

- Notre algorithme sélectionne la première tâche de la listetâches pour chaque processeur libre ou devient libre.
- chaque exécution d'une tâche on doit baisser la vitesse de processeur.
- Dans le cas d'arrivée d'une nouvelle tâche, on doit l'insérer dans la FAT et dans la position calculée par l'algorithme EDF*.
- Aucune tâche de la **FAT** n'est exécutée avant sa date d'arrivé, même si elle est la première dans **FAT** et il y a un processeur libre.
- Notre objectif est de maximiser le taux d'utilisation des processeurs, par la réduction des vitesses dans les cas possibles.
- La migration est assurée à l'aide d'utilisation d'une seule liste **FAT**, et l'affectation de la première tâche au premier processeur devenant libre.
- Dans le cas de préemption (aucun processeur libre et l'arrivé d'une tâche plus prioritaire que les tâches en cours d'exécution) on libère le processeur qui exécute la tâche de la plus grande laxité, et non pas obligatoirement le dernier processeur libre.
- Pour chaque événement si le nombre de tâches est inférieur ou égal au nombre de processeurs alors on exécute chaque tâche avec la vitesse minimale qui permet de respecter son échéance sans prendre en considération les autre tâches.
- les tâches doivent s'exécuter avant leur Deadline.
- Aucune préemption avant la terminaison de cycle processeur en cours d'exécution.

3.7 Etude de cas

Soit les 05 tâches périodiques T_1,T_2, T_3 , T_4, T_5, T_6 et T_7 tel que :

T_1 (R_1=1, C_1=3, Tex_1=2, D_1=4, P_1=1),

T_2 (R_2=1, C_2=2, Tex_2=3, D_2=10, P_2=1),

T_3 (R_3=2, C_3=2, Tex_3=2, D_3=11, P_3=1),

T_4 (R_4=2, C_4=3, Tex_4=2, D_4=5, P_4=1),

T_5 (R_5=2, C_5=2, Tex_5=2, D_5=12, P_5=1),

T_6 (R_6=3, C_6=3, Tex_6=2, D_6=6, P_6=1),

T_7 (R_7=3, C_7=4, Tex_7=2, D_7=8, P_7=1).

Vmax=1, Vmin=0.3, NBR_PROC=2

-A l'instant t=0 :

-la file d'attente est vide.

- A l'instant t=1 :

-La période d'étude est P=ppcm {1}=1

-les tâches T1, T_2 sont déjà arrivées.

-le nombre de tâches égal au nombre de processeurs alors on exécute chaque tâche avec la vitesse minimale qui permet de respecter son échéance sans prendre en considération les autres tâches.

-la tâches T1 exécuté dans le processeur 01 avec une vitesse min Vmin= 0.3, la tâches T2 exécuté dans le processeur 02 avec une vitesse Vmin= 0.3.

- A l'instant t=2 :

-le processeur01 est occupé (exécute T1) et n'a pas terminer l'exécution.

-les tâches T_3, T_4, T_5 sont déjà arrivées.

Après l'utilisation de notre technique DVS et l'algorithme EDF*, alors le nombre des tâches est supérieur au nombre des processeurs, alors la vitesse d'exécution Vmax.

- A l'instant t=3 :

-la tâche T_1 termine son exécution, et le processeur01 est libre.

-les tâches T6, T_7 sont déjà arrivées.

-la tâche T_3 exécuté dans le processeur 01

- A l'instant t=4 :

-la tâche T_2 termine son exécution.

-le processeur01 est occupé (exécute T3) et n'a pas terminer l'exécution.

-la tâche T_4 exécuté dans le processeur 02.

- A l'instant t=5 :

-la tâche T_3 termine son exécution.

-la tâche T_5 exécuté dans le processeur 01.

-le processeur02 est occupé (exécute T4) et n'a pas terminer l'exécution.

- A l'instant t=6 :

-la tâche T_4 termine son exécution.

-le processeur01 est occupé (exécute T5) et n'a pas terminer l'exécution.

-la tâche T_6 exécuté dans le processeur 02.

- A l'instant t=7 :

-la tâche T_5 termine son exécution.

-le processeur02 est occupé (exécute T6) et n'a pas terminer l'exécution.

-le nombre de tâches égal au nombre de processeurs alors on exécute chaque tâche avec la vitesse minimale qui permet de respecter son échéance sans prendre en considération les autres tâches.

-la tâches T_6 exécuté dans le processeur 02 avec une vitesse Vmin= 0.3.

- A l'instant t=8 :

-la file d'attente est vide.

-la tâche T_6 termine son exécution.

-le processeur01 est occupé (exécute T7) et n'a pas terminer l'exécution.

- A l'instant t=9 :

-la tâche T7 termine son exécution.

- A l'instant t=10 :

-la tâche T7 termine son exécution.

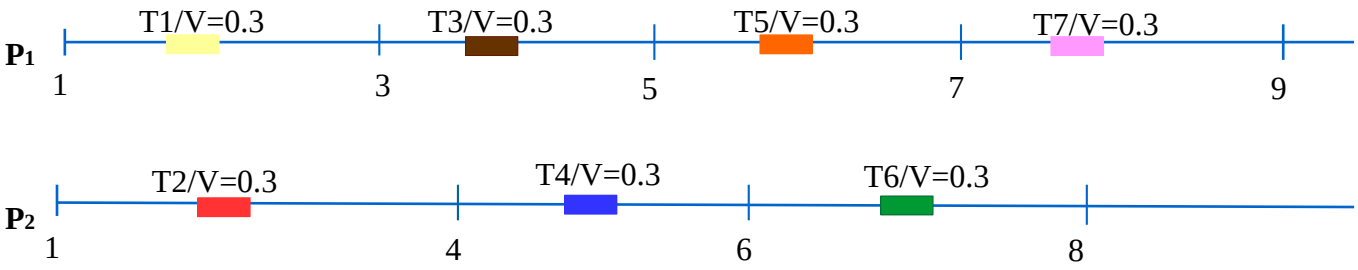

Figure 3.1 : Execution des tâches.

L'énergie totale consommée pendant une durée est calculée par l'intégration de la puissance, tel que la fonction de dissipation de puissance gi(S)= $a_i S^r$, au a_i>0 et r≥2 (a_i=1,r=2) est :

E_{tot}= $(0.3)^2$*3 + $(1)^2$*4 + $(0.3)^2$* 2+ $(1)^2$ *2+ $(0.3)^2$*2 + $(0.3)^2$*2+ $(0.3)^2$*2

E_{tot} = 0.27 + 4 + 0.18 + 2 + 0.18 + 0.18 + 0.18 = 6.99 joules.

Avec l'algorithme EDF seulement :

E_{tot}= $(1)^2$* 3 + $(1)^2$*2 + $(1)^2$*2 + $(1)^2$*2 + $(1)^2$*2 + $(1)^2$*3 + $(1)^2$*4

E_{tot}=3+2+2+2+2+3+4= 17 joules

Pour l'application de notre algorithme sur l'ensemble des tâches T_1, T_2, T_3, T_4, T_5, T_6 et T_7 on obtient un taux d'utilisation maximale de processeur de la date 0 à la date 10. Cet ordonnancement a une consommation optimale et permet à toutes les tâches de respecter leurs échéances.

4. IMPLEMENTATION

4.1 Présentation de logiciel

Notre logiciel est un outil d'aide à prendre la décision. Sa fonction principale est l'ordonnancement temps réel dans un système embarqué distribue. Et l'optimisation de coût énergique et temporel dans ce dernier.

4.2 Environnement de programmation

Notre logiciel a été développé en langage C++ Builder 6.0 pour les raison suivantes :

- Souplesse et puissance.
- Simplicité et portabilité.
- Il permet la modularité.
- Introduction de la notion de propriété événement.
- Communication direct avec le système d'exploitation.

4.3 Définition de l'environnement

C++ Builder est un environnement de développement basé sur C++, et ce dernier est un langage orienté objet permet la manipulation de classes, qui a été développé pour faciliter l'écriture et améliorer la qualité des logiciels en termes de modularité. Il est livré avec une bibliothèque de classes. Le développeur utilise ces classes pour mettre en point ses logiciels. C++ Builder est un outil RAD, c'est-à-dire tourné vers le développement rapide d'applications (Rapid Application Development) sous Windows. Donc C++ Builder permet de réaliser de façon très simple l'interface des applications et de relier aisément le code utilisateur aux événements Windows, quelle que soit leur origine (souris, clavier, événement système, etc.). Pour ce faire, C++ Builder repose sur un ensemble très complet de composants visuels prêts à l'emploi. La quasi-totalité des contrôles de Windows (boutons, boîtes de saisies, listes déroulantes, menus et autres barres d'outils) y sont représentés, regroupés par famille. Leurs caractéristiques sont éditables directement dans une fenêtre spéciale intitulée éditeur d'objets. L'autre volet de cette même fenêtre permet d'associer du code au contrôle sélectionné.

4.4 L'interface de C++ Builder

La figure 1 représente un exemple typique de l'interface de C++ builder au cours de session de travail.

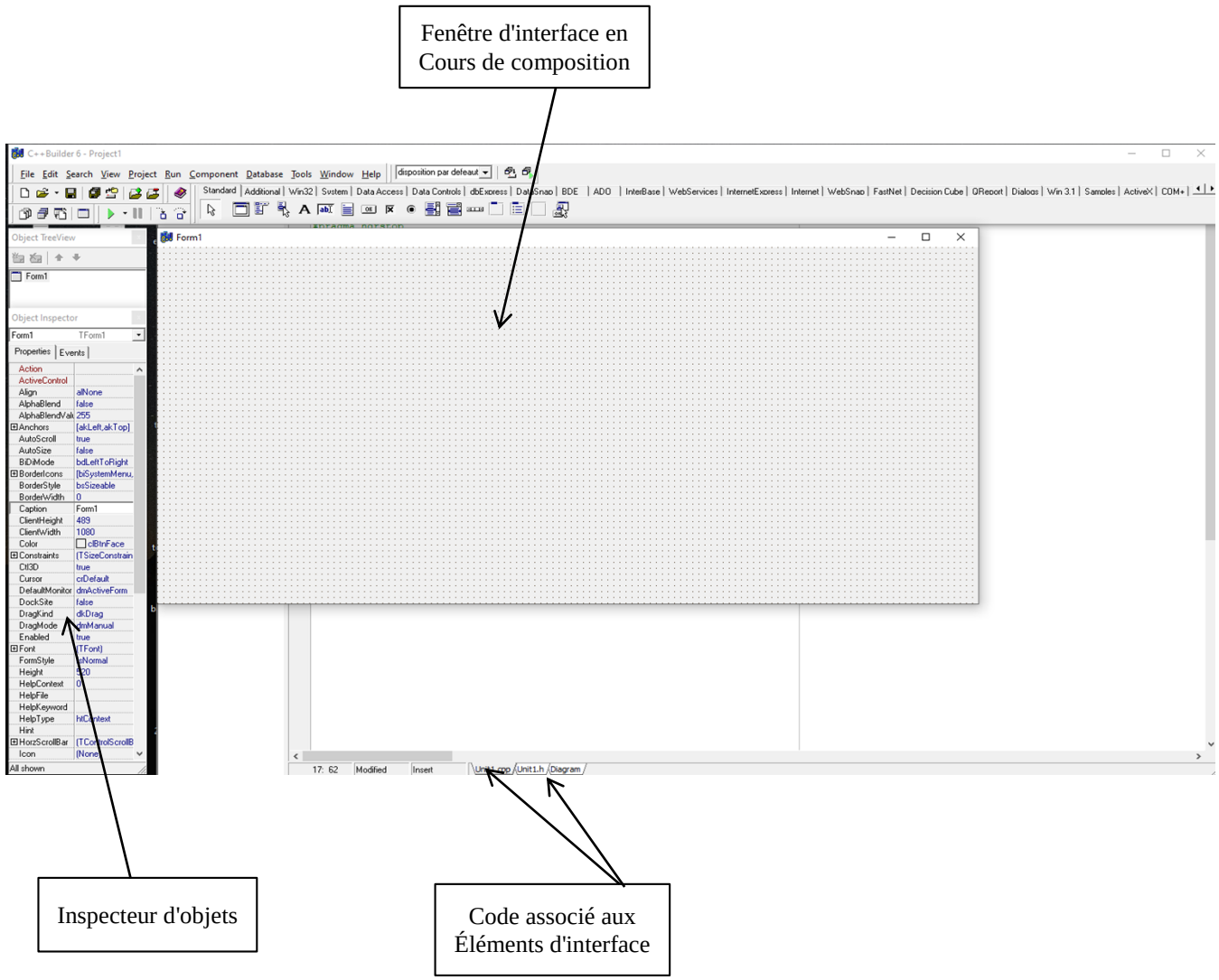

Figure 3.2: interface de C++ Builder

4.5 Les composantes de C++ Builder

Par défaut, C++ Builder utilise un compilateur C++, un éditeur de liens, un compilateur de ressources et un gestionnaire de projets intégrés. Il est toutefois possible de spécifier que vous désirez lui faire utiliser les outils en ligne de commande livrés conjointement ou même d'autres outils. Ce dernier cas, très intéressant lorsque l'on souhaite utiliser des modules compilés dans d'autres langages (c'est tout même un peu technique) doit être étudié très soigneusement. En particulier, il faut s'assurer que les différents codes soient compatibles.

Théoriquement, C++ Builder se charge de gérer les imports et les exports des différentes librairies dynamiques (DLL) utilisées. Toutefois, on peut gérer cela manuellement (notamment pour éviter qu'une propre DLL n'exporte toutes ses définitions, ce qui est le comportement par défaut) en éditant manuellement le fichier des définitions (.DEF) puis en appelant l'utilitaire emplib.

4.5 Interface de logiciel

Notre logiciel est caractérise par son interface graphique simple et significative qui facilite son utilisation, qui contient deux fenêtres tels que:

-la première fenêtre appeler création tâches qui permet à l'utilisateur d'ajouter, de modifier, de supprimer ou de valider une tâche, qui est affichée dans une table de base de données sous Access 2007 connectée au programme à l'aide d'outils c++.

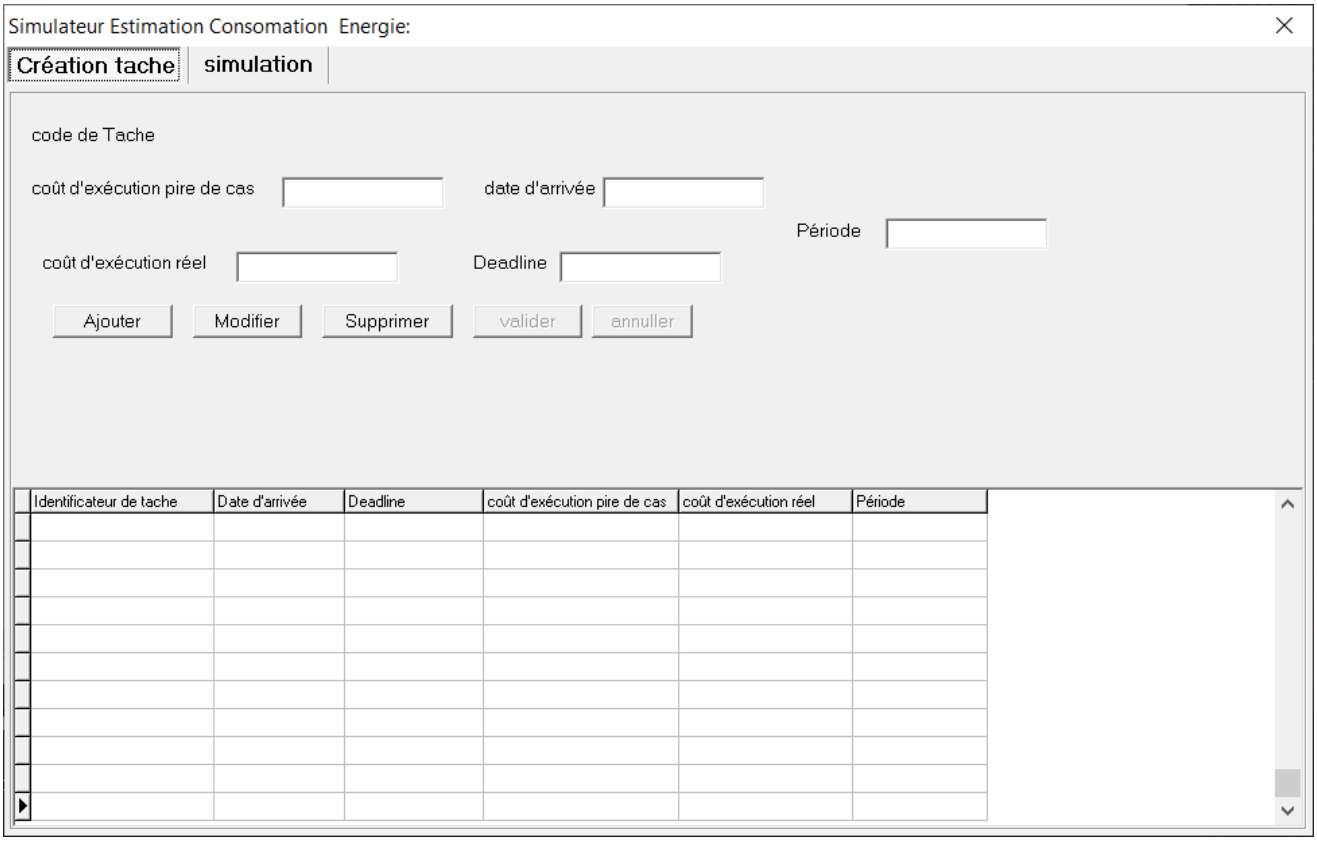

Figure 3.3: interface de la création des tâches.

-La deuxième fenêtre appelée Simulation permet à l'utilisateur de choisir dynamiquement le nombre De processeurs pour exécuter ses tâches. Et en bas se trouvent les tâches qui attendent l'exécution dans les processeurs.

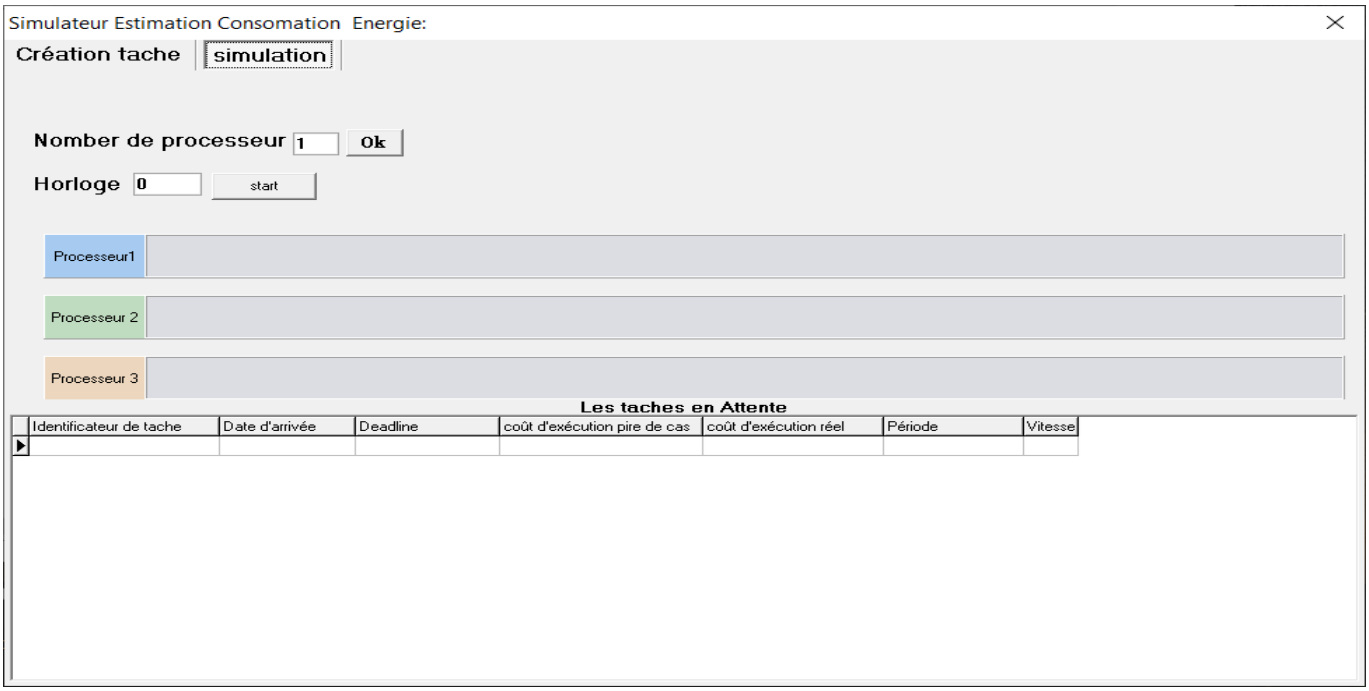

Figure 3.4: interface de la création des tâches.

5. Exemple d'exécution

Voilà un exemple d’exécution de notre solution :

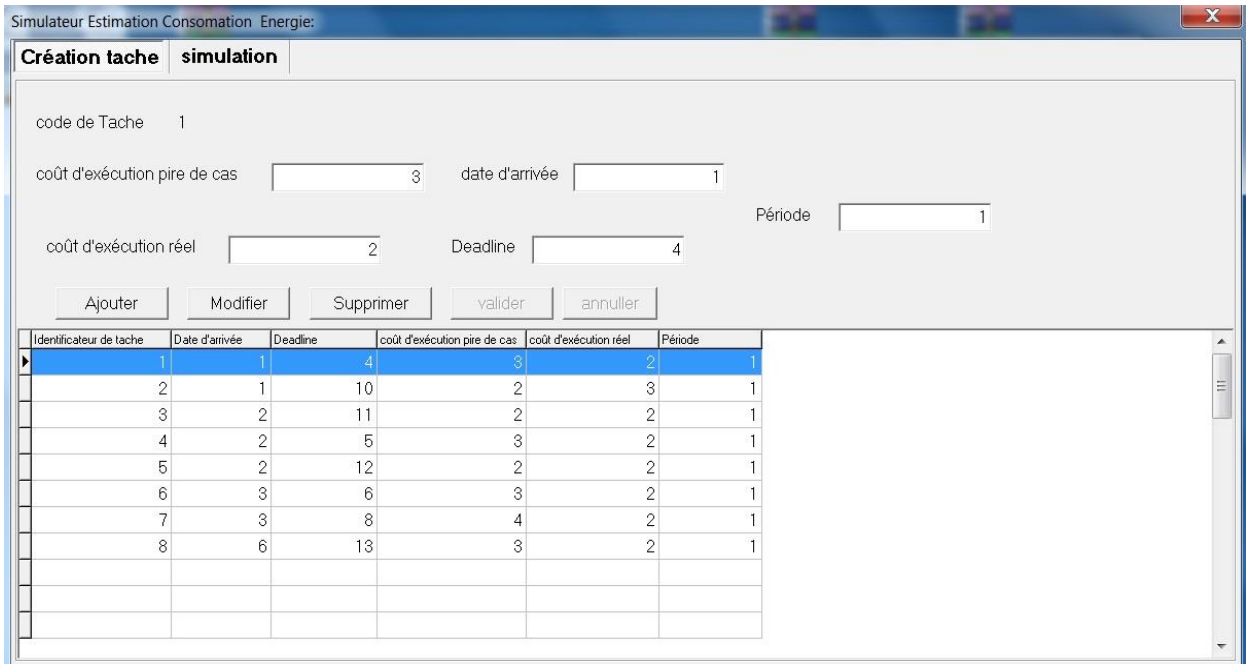

Figure 3.5 : L'exécution de notre model

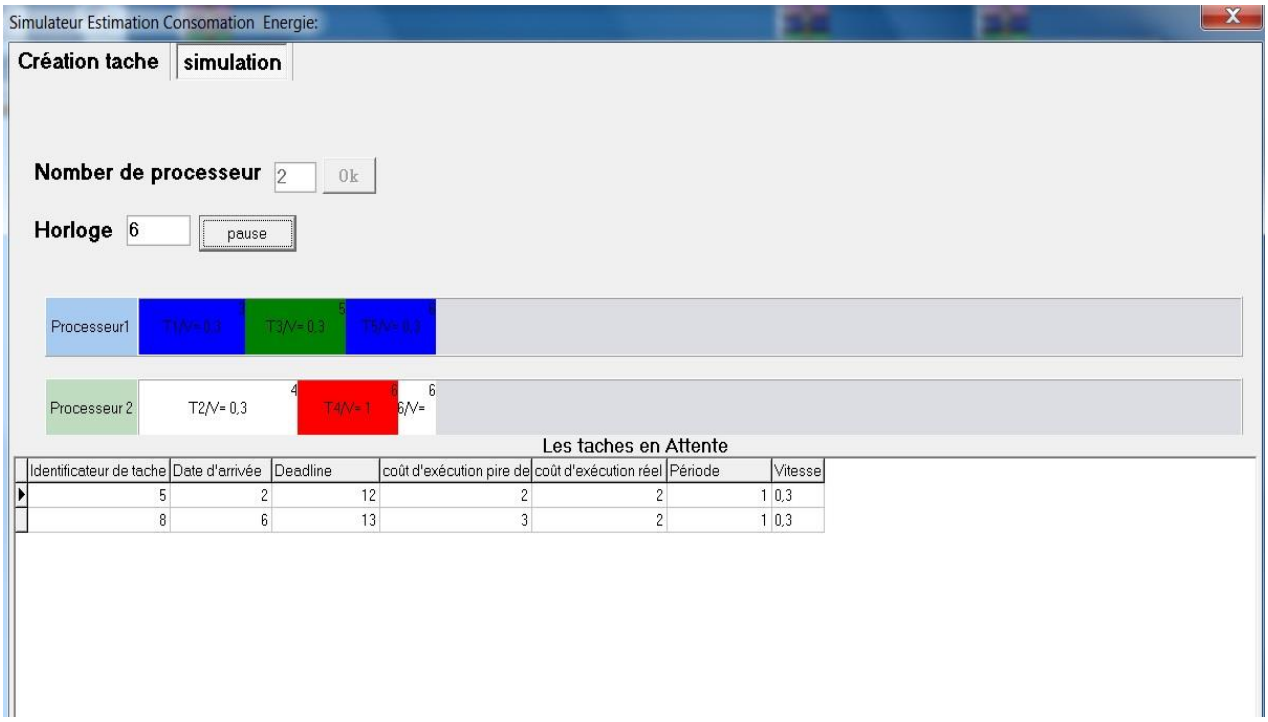

Figure 3.6 : la suite d'exécution

Conclusion

Tous les algorithmes d'ordonnancement temps réel pratiques dans les systèmes distribués présentent un compromis entre la performance et de la complexité de calcul. Dans les systèmes distribués temps réel, les tâches doivent être exécutées en temps opportun. La recherche sur l'ordonnancement des tâches a principalement mis l'accent sur le temps de calcul, le laxisme, la priorité, la consommation d'énergie etc. L'intégration de la théorie floue dans les algorithmes d'ordonnancement peut vraiment faire l'étude très intéressante. Trouver un ordonnancement optimal dans les systèmes distribués, avec des contraintes temps réel s'avère NP-difficile. L'incertitude intrinsèque dans les systèmes dynamiques en temps réel augmente les difficultés d'algorithmes d'ordonnancement. Pour cela, nous avons proposé une approche d'ordonnancement floue pour organiser les tâches périodiques en temps réel par rapport à la consommation optimale d'énergie sur une architecture multiprocesseurs. La pièce actuelle de la recherche a été simulée sur c++ builder 6.

Les résultats obtenus mettent en évidence le comportement de notre travail et montre l'efficacité de l'utilisation de l'algorithme EDF pour ce problème.

Conclusion
et perspectives

Conclusion et perspectives

Nous avons présenté dans ce mémoire, notre nouvelle solution au problème de l'ordonnancement dynamique d'ensembles de tâches périodique indépendants, définies sous contraintes temporelle. La solution proposée se focalise sur l'application de la technique dite : Mise à l'échelle (adaptation) de tension et de fréquence (DVS pour Dynamic Voltage Scaling) au problème d'ordonnancement temps réel prenant en compte l'optimisation de la consommation d'énergie au niveau des systèmes embarqués temps réel.

Nous pensons que l'algorithme EDF et la technique DVS peuvent trouver des solutions proches de l'optimal surtout pour les problèmes d'optimisation de consommation d'énergie. Ce constat est justifié par le fait qu'on traite des informations imprécises et incertaines. Ces informations englobent par exemple les temps d'arrivées des tâches, les temps d'exécutions réels des tâches qui sont généralement très loin de leurs temps d'exécution au pire de cas, la meilleure fréquence d'horloge du processeur qui mène à la consommation minimale, le bon moment pour migrer
une tâche vers un autre processeur, etc. L'incertitude intrinsèque dans les systèmes temps réel et en particulier les systèmes dynamiques augmente les difficultés d'algorithmes d'ordonnancement conventionnels pour optimiser la consommation d'énergie. En intégrant la technique DVS dans le problème d'ordonnancement temps réel, les décisions de l'ordonnanceur vis-à-vis de choix de la meilleure fréquence d'horloge du processeur, de priorités et des dates de migration des tâches, peuvent être améliorées considérablement

Cette recherche vise à répondre aux besoins des nouvelles applications temps-réel dans les systèmes à temps réel, pour lesquelles la consommation d'énergie joue un rôle très important. Dans ce contexte, nous avons présenté les résultats théoriques de l'algorithme pour le cas multiprocesseur. Les objectifs à minimiser sont : l'énergie, le temps moyen de réponse. Nous avons effectué des simulations sous c++ builder dans le but est d'évaluer le comportement de l'algorithme. Ces simulations ont notamment confirmé le très bon comportement de l'algorithme proposé en termes de consommation d'énergie vis-à-vis des tâches périodiques indépendantes. Toutes les étapes de la conception ont été développées dans le troisième chapitre.

Ce mémoire traitant uniquement l'ordonnancement des tâches périodique indépendantes sur une architecture monoprocesseur et multiprocesseur sous contraintes temporelles.
Les perspectives envisageables sont :
1.Prise en compte des tâches apériodiques.

2.Explorer d'autres techniques de croisement et de sélection.

3.Comparer notre solution avec d'autres solutions qui sont basées sur des heuristiques d'optimisation différentes.

Bibliographie

Bibliographie :

[ACD 17] Anthony Canino et Yu David Liu, «Proactive and adaptive energy-aware programming with mixed typechecking», PLDI 2017 Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM, 18-23 Juin 2017, p.217-232.

[ALB 07] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. ACM Trans. Algorithms, 3, November 2007.

[BOU 12] BOUNABI Chafia, Optimisation des performances dans les systèmes embarqués distribué, mémoire de fin d'étude 2012/2013.

[BOU 18] Samia BOUZEFRANE, Maître de Conférences CEDRIC CNAM, Samia.bouzefrane@cnam.fr

[COG 19] Coggle, diagramme des systèmes embarqués, 2019.

[DAN 04] Dana - Mihaela ROHÁRIK VÎLCU, Ordonnancement optimal de tâches pour la consommation énergétique du processeur, thèse doctorat, Université Paris XII – Val de Marne, 2004.

[HEI 00] Heinzelman, W.R., A. Chandrakasan, & H. Balakrishan. Energy-efficient Communication Protocol for Wireless Microsensor Networks. Proc. of the 33rd Hawaii International Conference on System Science. IEEE Press. (pp. 3722-3725) 2000.

[KAD 05] KADIONIK Patrice, ENSEIRB Les Systèmes embarqués. Linux embarqué, 2005.

[KER 09] Omar KERMIA, Thèse présentée pour obtenir le grade de Docteur en science : Ordonnancement temps réel multiprocesseur de tâches non-préemptives avec contraintes de précédence, de périodicité stricte et de latence, 19/03/2009.

[KIM 02] Woonseok Kim, Dongkun Shin, Han-Saem Yun, Jihong Kim, and Sang Lyul Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. InProceedings of the 8th IEEE Real-Time and Embedded Technol-ogy and Applications Symposium (RTAS'02), San Jose, California, page 219–228, September 2002.

[KOP 13] Hermann KOPTEZ, Real-Time Systems Design Principles for Distributed Embedded Applications, Second Edition, 2013

[LAU 18] LAURENT Pautet, cours ordonnancement temp réel, Laurent.Pautet@enst.fr

[LEG 14] Vincent LEGOUT, Ordonnancement temps réel multiprocesseur pour la réduction de la consommation énergétique des systèmes embarqués. Calcul parallèle, distribué et partagé. Télécom ParisTech, 08 Avril 2014.

[MAN17] MANSOURI Khalifa, Enseignant Chercheur at ENSET, Université Hassan II de Casablanca, Cours systèmes temps réel, Novembre 11, 2017.

[MEH 14] MEHALAINE Ridha, Optimisation de la consommation d'énergie sous contraintes temporelles au niveau des systèmes embarqués distribués temps réel, mémoire de magister 2014/2015.

[MES 11] MESSAOUDI Fatima, Estimation de la consommation d'énergie dans les systèmes embarqués, 2011-2012.

[MES 19] E. Messerli (HES-SO / HEIG-VD / REDS), systèmes embarqués, 2019.

[MNW01] M.Kandemir, N.Vijaykrishnan, M. J.Irwin et Wu Ye, «Influence of Compiler Optimizations on System Power», IEEE Transactions on Very Large Scale Integration (VLSI) Systems, IEEE, vol.9, no 6, December 2001, p.801-804.

[NAG 06] Nicolas NAVET et Bruno GAUJAL, Ordonnancement temps réel et minimisation de la consommation d'énergie, 2006.

[OPT 19] Optima Junior Entreprise, historiques des, 7 Janvier 2019

[PRE 00] Jane W. S. Liu, "Real-time Systems ", Prentice Hall, 2000.

[QJY 14] Qianao Ju et Ying Zhang, «Reducing Charge Redistribution Loss for Supercapacitor-Operated Energy Harvesting Wireless Sensor Nodes», ENSsys '14 Proceedings of the 2nd International Workshop on Energy Neutral Sensing Systems, ACM, 6 November 2014, p.31-36.

[RAM 17] Ramzy Rammouz. Optimisation de la gestion d'énergie dans les systèmes embarqués. Réseaux et télécommunications. Université de Lyon; Université libanaise, 2017. Français.

[SAO 03] Saowanee Saewong and Ragunathan Rajkumar. Practical voltage-scaling for fixedpriority RT-systems. InProceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Washington, DC, pages 106–114, May 2003.

[SMI 97] Smith, J.S.SApplication Specific Integrated Circuits. Addision Wesley, (1997).

[TEF 14] TEFY Lory, Types et conduite sécuritaire d'un automatisme portail, 13 Mai 2014.

[UGE 19] Akram BEN AISSI, système d'exploitation embarqué, université GustavE

Eiffel, 2019.

[WIK 19] Ordonnancement dans les systèmes d'exploitation, Un article de Wikipédia, 2019.

[WWD94] Weiser (Mark), Welch (Brent), Demers (Alan) et Shenker (Scott). Scheduling for reduced CPU energy. In: Proceedings of the First Symposiumon Operating System Design and Implementation. Novembre 1994.

[ZLX 15] Zimeng Zhou, Lei Ju, Zhiping Jia et Xin Li, « Managing Hybrid On-chip Scratchpad and Cache Memories for Multi-tasking Embedded Systems », Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific, IEEE, 19-22 janvier 2015,p.423-428.

[ZUI 16] Stéphan Van ZUIJLEN, informatique embarqué et objets connectés, 2016.