Ordonnancement Temps Réel: Synthèse des Théories et Analyse de Leurs Défauts

Claude Kaiser

Conservatoire National des Arts et Métiers
Laboratoire CEDRIC
292 Rue Saint Martin,
75141 Paris Cedex 03
FRANCE

Joëlle Delacroix

Université Pierre et Marie Curie
Laboratoire MASI
4 place Jussieu,
75252 Paris Cedex 05
FRANCE

Abstract: The paper accounts for the authors'endeavour in bringing to light the basic researches carried out in the field of real-time job scheduling.Their findings suggest the apartness of basic research and applied research in the field of real-time scheduling. The consequences ensued by such an apartness and the methods to be used in preventing it are the authors' main concerns. The methods make use of an importance criterion thereby job scheduling can properly consider time constraints. A critical approach of these recent concepts will allow improved solutions to be adopted.
Keywords: Real-time,time constraints observance,time fault, non- stability, importance criterion.

Résumé: Dans cet article, nous nous proposons d'analyser la manière dont le problème spécifique de l'ordonnancement des tâches en temps réel est abordé dans les théories développées par les chercheurs.
Nous verrons que l'approche réalisée par les théories est en fait distante du problème tel qu'il se pose aux concepteurs d'applications. Nous présentons alors les conséquences engendrées par le décalage existant, puis nous discutons les méthodes qui jusqu'alors ont été proposées pour le combler. Ces méthodes se basent sur l'emploi d'un critère d'importance permettant, à tout instant, d'assurer l'ordonnancement des tâches les plus importantes de l'application et ce dans le respect de leurs contraintes de temps. Nous discutons ces nouvelles théories et dégageons de cette critique, des axes souhaitables d'amélioration.
Mots-Clés: Temps réel, respect des contraintes temporelles, faute temporelle, instabilité, critère d'importance.

Claude Kaiser is professor of Computer Operating Systems and Programming at Conservatoire National des Arts & Métiers (CNAM),Paris,France and chairman of the Computer Science Department.CNAM is a Technological University providing courses for continuing education of professionals and technicians.
Claude Kaiser is co-author of several books on operating systems or on programming. He participated in the design of real-time systems and of time-sharing operating systems when he was working for the French Navy and when he was senior scientist at Institut National de Recherche en Informatique et en Automatique (INRIA). He has been contributing as a Scientific Advisor to projects such as the redundant system Saphir, the Chorus Distributed Operating System and the Dune IX Real-Time Operating System. His current interest is in distributed algorithms, in real- time schedulers and in the use of ADA for the teaching of Operating Systems.

Claude Kaiser est professeur d'informatique, en programmation et en systèmes, au CNAM, Paris,France et y est président du Département d'informatique. Le CNAM (Conservatoire National des Arts et Métiers) remplit les fonctions d'Université Technologique pour la formation permanente des professionnels et des techniciens.
Claude Kaiser est le co-auteur de plusieurs livres publiés sous un nom collectif pour les systèmes (Crocus,Cornafion) ou la programmation (Grégoire). Il participa à la conception d'un système temps réel pour la Marine et d'un système en temps partagé pour l'INRIA. Il a été conseiller scientifique pour des projets comme le système doublé Saphir pour Cerci, le système distribué Chorus et le système temps réel Dune IX. Il s'intéresse actuellement plus particulièrement à l'utilisation du langage ADA comme support de l'enseignement des systèmes, à l'algorithmique distribuée et a l'ordonnancement des systèmes informatiques temps réel.

Joëlle Delacroix was born in Clichy,France,in 1967. She studied Computer Science at " Pierre et Marie Curie" University in Paris.In 1991 she took her high-specialisation diploma in computer systems. Currently, she is preparing, under the co-ordination of Professor Claude Kaiser from Conservatoire National des Arts & Métiers, a thesis on the scheduling of centralized and distributed real-time systems.

Joëlle Delacroix est née en 1967 à Clichy, France. Après des études en informatique a l'Université Pierre et Marie Curie de Paris, elle a obtenu en 1991 le diplôme d'études approfondies (DEA) de Systèmes Informatiques de cette université. Elle achève actuellement, sous la direction du professeur Claude Kaiser du Conservatoire National des Arts et Métiers, une thèse portant sur l'ordonnancement dans les systèmes temps réel centralisés et répartis.

1. Introduction

Peut être qualifiée de "temps réel", toute application mettant en oeuvre un système

informatique dont le fonctionnement est assujetti à l'évolution dynamique de l'état d'un environnement (le procédé) qui lui est connecté et dont il doit contrôler le comportement [1].

Les applications temps réel couvrent un grand nombre de domaines comme le pilotage d'usines robotisées, l'aérospatiale (télésurveillance de la fusée Ariane, contrôle de satellites et de stations spatiales), l'exploration sous-marine, le nucléaire (surveillance des réacteurs), le transport (aide à la conduite du TGV, pilotage du VAL), les télécommunications et le militaire.

Par exemple, dans une usine robotisée, le procédé est constitué de la chaîne de montage avec les robots, les points d'assemblages et les pièces manipulées. Dans une centrale nucléaire, il est formé des réacteurs et de leurs mécanismes de régulation. Le système de contrôle est composé de l'ordinateur qui gère et coordonne les activités sur la chaîne ou déclenche les mécanismes de régulation d'un réacteur.

L'interface entre le procédé et le système de contrôle est constituée par un ensemble de capteurs et d'actionneurs qui respectivement, permettent au procédé de délivrer une image de son état au système de contrôle et à celui-ci de piloter cet état (Figure 1).

Le système de contrôle suit donc et pilote l'état du procédé, en réagissant à toute évolution jugée significative. Ses actions ont par conséquent des instants d'exécution directement ou indirectement assujettis aux signaux émis par le procédé (événements) ou prélevés (mesures) sur le procédé. Ce pilotage, en outre, doit assurer que le procédé reste toujours totalement contrôlé, notamment à cause de l'impact physique et humaine que les décisions du système de contrôle ont sur le procédé. Par exemple, l'absence ou un retard de réaction au dysfonctionnement d'un robot sur la chaîne de montage peut aboutir a la détérioration de celle-ci, ou tout du moins d'une partie des pièces assemblées.

Ainsi, dans un système temps réel, toutes les contraintes portant sur les ressources et la correction du comportement, sont combinées à une contrainte de temps: la réaction du système de contrôle à un signal émis par le procédé doit intervenir dans un temps de réponse imposé ou recommandé qui définit une **échéance**. Le degré de contrainte sur le temps de réponse dépend de la nature du procédé et des conséquences que peut entraîner le non- respect de ce temps. Il convient ici d'attirer l'attention du lecteur sur une confusion trop généralement faite sur ce qu'est le temps réel. On voit qu'il ne s'agit pas d'aller vite, mais simplement de respecter le délai imposé pour réagir aux événements survenus dans le procédé. En somme, que le délai soit de deux heures ou d' une seconde ne change rien. La

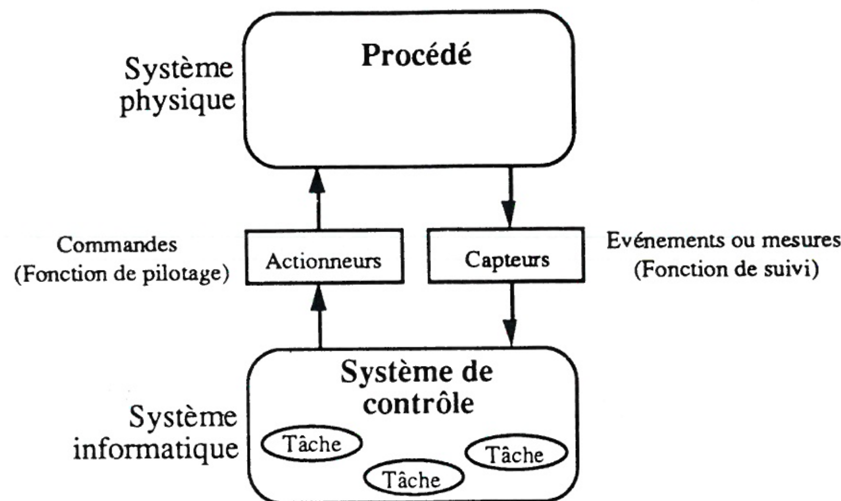


Figure 1. Environnement temps réel

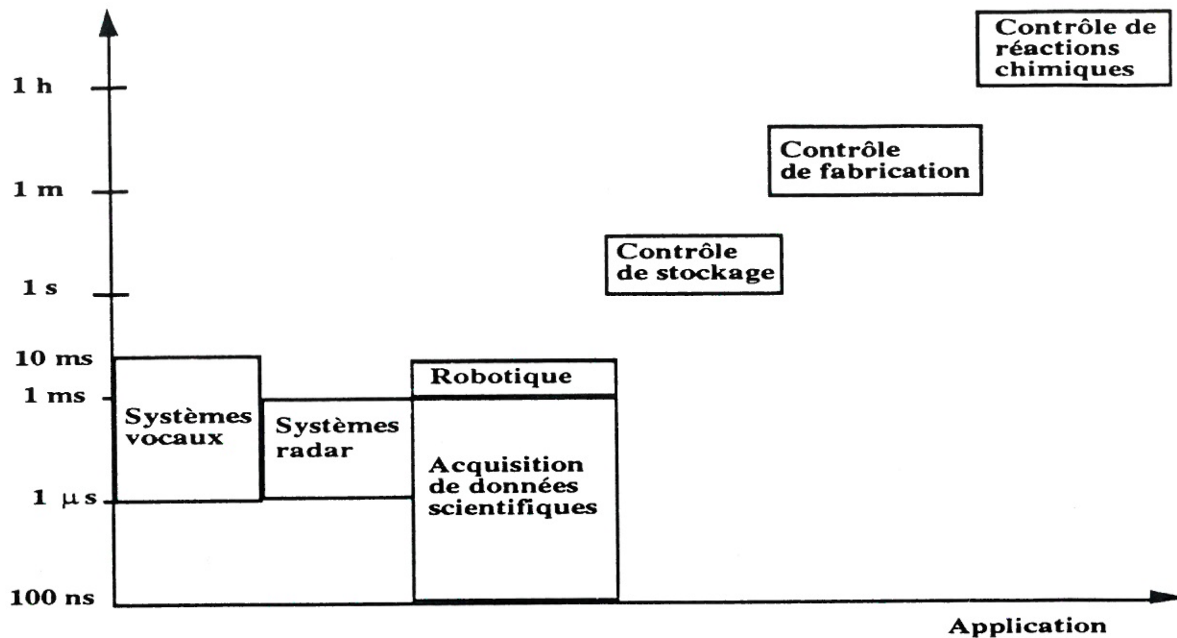


Figure 2. Applications et contraintes temporelles

Figure 2 donne un panorama des contraintes en fonction du type d'applications.

Les tâches composant une application temps réel peuvent donc être, en fonction de leurs caractéristiques, répertoriées sous différents types. Deux critères, évoqués ci-dessus, interviennent principalement:

- le degré de **flexibilité** de la tâche vis-à-vis de l'application temps réel, i.e. son "droit" à dépasser son échéance. On distingue les **tâches à contraintes strictes (tâches critiques)**, qui ne doivent jamais violer leur échéance sous peine de mettre le procédé en péril et les **tâches à contraintes relatives (tâches essentielles)**, qui elles, peuvent de temps à autre manquer leur échéance (avec une certaine amplitude) sans que le système temps réel en souffre.

  Lorsqu'une tâche dépasse son échéance, nous dirons qu'il se produit une **faute temporelle.**

  Exemple de tâche à contraintes strictes: maintien des équipements électroniques d'un satellite dans des plages de température évitant leur détérioration.

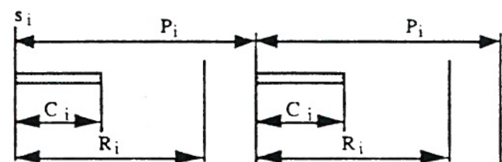


Figure 3. Tâche périodique

  Exemple de tâche à contraintes relatives: maintien de l'antenne du satellite exactement pointée vers la Terre. Un léger retard entraîne seulement une perte de puissance; cependant, un retard plus important peut la déstabiliser.

- la façon dont la tâche intervient dans la vie du système temps réel. On distingue les **tâches périodiques,** qui s'exécutent à intervalles réguliers (typiquement des mesures ou les commandes de systèmes asservis échantillonnés) et les **tâches**

apériodiques qui surviennent aléatoirement (typiquement une alarme).

Dans une application, l'ensemble des tâches périodiques se charge de la conduite générale du procédé. Les tâches apériodiques sont elles dévolues au traitement d'événements asynchrones particuliers comme les erreurs, les pannes... L'occurrence de tâches apériodiques peut entraîner une surcharge durant laquelle il devient difficile de respecter toutes les contraintes de temps.

L'ordonnanceur d'un système temps réel travaille donc avec quatre types de tâches:

- des tâches périodiques à contraintes strictes et des tâches périodiques à contraintes relatives (Figure 3).
  Elles sont définies par le quadruplet T_i (s_i, C_i, R_i, P_i) où
- s_i est la **date** de première requête de la tâche. Les requêtes suivantes ont lieu à $t = s_i + kP_i$. Si ($s_i = s_j$ pour tout i,j), les tâches sont dites **à départ simultané**[1] sinon elles sont dites **à départ échelonné**[1].
- C_i, P_i et R_i sont des durées: respectivement le **temps d'exécution** de la tâche, la **période** de la tâche, le **délai critique** de la tâche. Celle-ci doit avoir achevé l'exécution de sa $k^{ième}$ requête à la date d'échéance $d_i = s_i + kP_i + R_i$. Si l'échéance d'une requête correspond à la date de réveil de la requête suivante (on a $d_i = s_i + (k+1)P_i$) alors la tâche est dite **à échéance sur requête.**
- des tâches apériodiques à contraintes strictes et des tâches apériodiques à contraintes relatives (Figure 4):

Figure 4. Tâche apériodique

Elles sont définies par le triplet R_i (r_i, C_i, d_i) où

- r_i est la **date** d'arrivée de la tâche.
- C_i est la **durée** d'exécution de la tâche.
- d_i est la **date** d'échéance de la tâche.

2. Le problème de l'ordonnancement en temps réel

Nous allons définir le problème de l'ordonnancement en temps réel, puis examiner la façon dont les théories l'abordent.

2.1. Définition du problème

2.1.1 Contexte

Le contexte dans lequel le problème se pose est le suivant:

- L'application peut être constituée de plusieurs phases.
  A un instant t de la vie de l'application, toutes les tâches de celle-ci ne sont pas susceptibles d'être exécutées. Le sous- ensemble des tâches exécutables à t et durant un temps Δt définit une phase de l'application. Le basculement de l'application d'une phase à l'autre est provoqué par l'occurrence d'un événement donné, par le réveil ou l'activation d'une tâche et il peut entraîner le réveil, l'activation ou la mise en sommeil d'autres tâches.

1 Les termes tâches synchrones et tâches asynchrones sont plus généralement utilisés, cependant ces adjectifs sont également employés pour qualifier des exécutions de tâches ayant des périodes multiples les unes des autres et sans plus de précision, les expressions tâches synchrones et tâches asynchrones sont ambiguës.

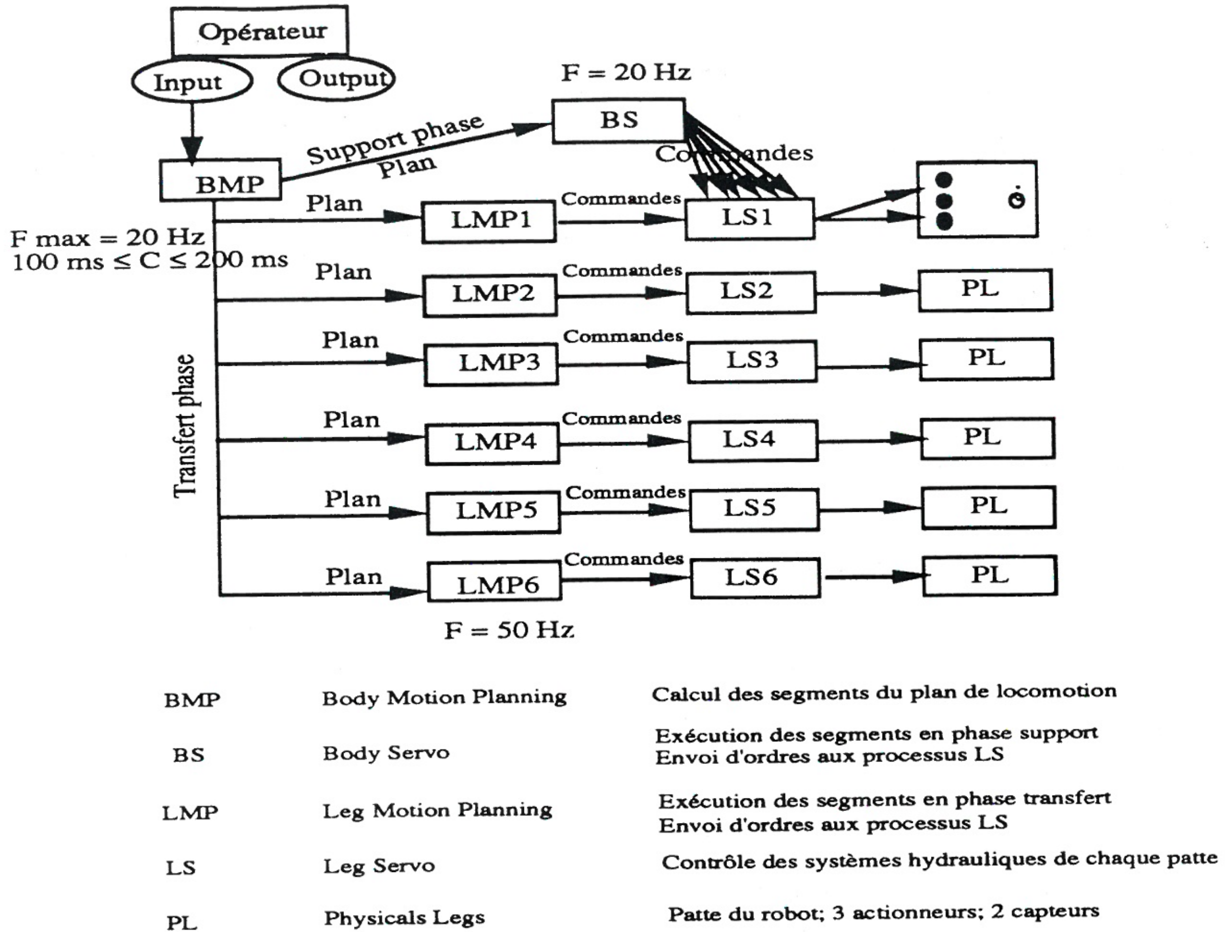


Figure 5. Application ASV

Exemples:

- 1/Application Robotique ASV (Adaptative Suspension Vehicle [2] [3] (Figure 5): l'application comprend une phase dite Transfert Phase lorsque la patte du robot est en mouvement dans les airs et une phase Support Phase lorsque la patte du robot est sur le sol.
  Lors de la phase Transfert, c'est la tâche LMP (Leg Motion Planning) qui est active alors que pendant de la phase Support, c'est la tâche BS (Body Servo). Ces deux modes sont activés par la tâche construisant la trajectoire à suivre (tâche BMP - Body Motion Planning) selon ses besoins. La longueur de chaque phase est indéterminée et dépend du traitement (segment directif) à exécuter.
  La tâche BMP appartient, elle, aux deux phases de l'application.
- 2/Application de télésurveillance: Surveillance de jour - Surveillance de nuit.
  Les conditions de travail dans chacune des phases sont différentes et le matériel à piloter également (caméra infra- rouge la nuit). Par ailleurs, on peut imaginer que la surveillance de nuit est plus sévère que celle de jour et utilise donc des moyens supplémentaires (surveillance sonore, surveillance de zones supplémentaires ignorées le jour). Le passage d'une phase à l'autre peut être effectuée par une tâche testant la luminosité ambiante ou à une heure pré-programmée.

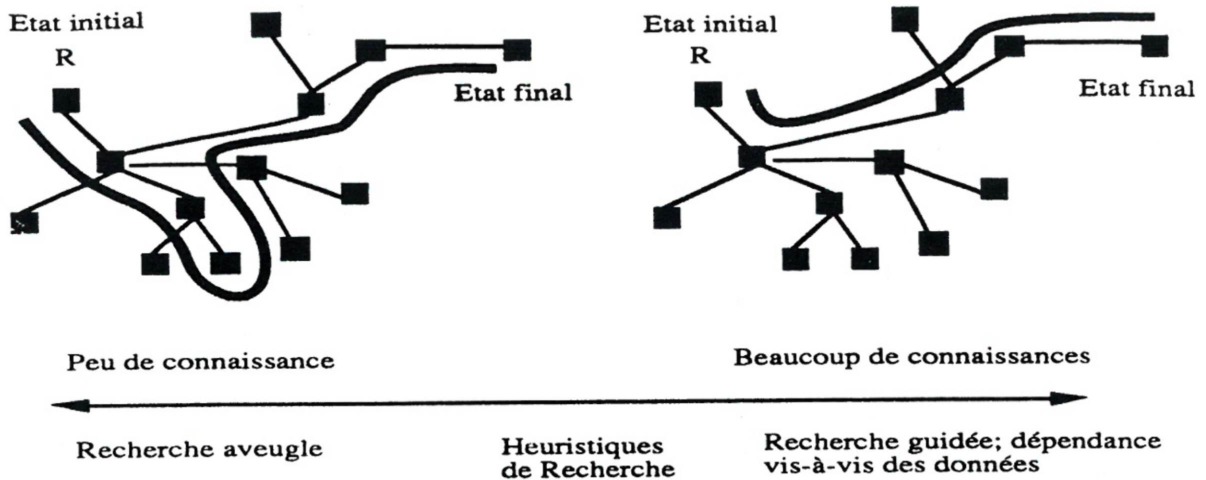


Figure 6. Variation de temps d'exécution pour une recherche

- Les tâches requièrent des ressources, le processeur bien-sûr, mais aussi des périphériques comme des imprimantes, des disques, des fichiers, des voies de communication, etc.
- Les tâches sont caractérisées par un paramètre **Importance** qui définit leur poids relatif dans l'application, c'est-à-dire la façon dont leur exécution est cruciale vis-à-vis du maintien du procédé dans un état sécuritaire.
- Les paramètres temporels des tâches et notamment le temps d'exécution et la période peuvent être variables.

Parce que souvent on ne connait pas toutes les caractéristiques de l'environnement et qu'on ne peut prévoir l'évolution de celui- ci, le comportement de nombreuses applications demeure incertain et ne peut pas être prédéterminé avec exactitude. Prenons, par exemple, dans un contexte futuriste, toutes les applications du domaine de l'exploration spatiale (sondes expédiées dans le système solaire, exploration robotique des planètes voisines de la Terre [4]): elles constituent des exemples pour lesquels la taille, la complexité, la longueur de vie du système temps réel, la non-maîtrise de l'environnement dérangent la perception de ce qui peut advenir: les temps d'exécution sont alors aléatoires et le pire temps d'exécution d'une tâche est largement supérieur au temps moyen d'exécution de la tâche ou bien incalculable. De même, les fréquences d'exécution des tâches peuvent évoluer en fonction de l'état des propriétés et du comportement de l'environnement dans lequel l'application progresse.

Analyse des causes de variation

Deux grandes classes se distinguent, la source des variations des paramètres temporels pouvant être soit l'application temps réel elle-même, soit le système et la machine hôte.

- **Les variations inhérentes à l'application**

Nous ne prétendons pas en faire une liste exhaustive, mais simplement en donner quelques exemples.

Les variations peuvent provenir soit du passage dans le code de l'application (nombre d'exécutions des boucles, branche conditionnelle), soit de la quantité et de la nature des données à traiter.

Exemples

* Soit un processus périodique pilotant et contrôlant la température d'un périphérique. Son temps d'exécution sera vraisemblablement court si la température prélevée est normale (juste un test pour vérifier que la température est dans les bornes acceptables), par contre, elle risque d'augmenter en cas d'anomalie (en plus du test, viennent les actions de correction). De même, on peut imaginer qu'à partir du moment où une anomalie est détectée, la fréquence d'activation de la tâche de contrôle augmente afin de mieux surveiller le rétablissement des propriétés normales de l'environnement.
* Prenons le Serveur Périodique de tâches apériodiques [5]. Son temps d'exécution varie évidemment en fonction du nombre de tâches apériodiques à servir.
* L'application Adaptative Suspension Vehicle (ASV) [3] [2] fournit des exemples de variations tant au niveau des temps d'exécution que des périodes des tâches. La tâche Body-Motion- Planning détermine le mouvement de chacune des six pattes du robot en planifiant la trajectoire (plan de locomotion) à suivre sous forme de segments successifs calculés périodiquement et ajoutés au plan de locomotion. Cette tâche s'exécute à des fréquences différentes tout au long de l'application, selon l'état du robot, la nature du terrain sur lequel il évolue et selon les ordres transmis par l'opérateur. Ainsi, au démarrage du robot ou lorsque le lieu parcouru est escarpé, les segments calculés sont courts et en conséquence, la tâche est exécutée plus fréquemment; la vitesse de croisière atteinte, le sol aplani, les segments sont allongés et la fréquence d'exécution est diminuée.
* Un dernier exemple est fourni dans [6] pour le domaine des applications temps réel en l'Intelligence Artificielle. Le modèle général de simulation de l'intelligence humaine est celui d'une recherche-comparaison dans un ensemble de solutions possibles. Cette recherche peut être plus ou moins longue selon la somme de connaissances dont on dispose pour l'orienter (Figure 6). Dans un environnement totalement caractérisé, la recherche est complètement dirigée et les variations proviennent seulement de la nature des données. Cependant, moins on dispose de connaissances, plus la recherche devient aveugle et plus l'ampleur des variations des temps d'exécution devient importante.

  Ce type de problème se pose également lorsque l'application temps réel travaille avec une base de données; le temps d'exécution des transactions varie en fonction de la nature de la requête et surtout du nombre de relations ou de fichiers à traverser.

- **Les variations inhérentes au système ou au matériel**

Au niveau du matériel, les composants électroniques peuvent subir des perturbations passagères (radioactivité) ou permanentes (dégradation physique) qui affectent les performances de la machine et ralentissent son rythme de fonctionnement.

Au niveau du système, les arrivées d'interruptions internes (IT horloge, trappe pour défaut de pages) ou externes (lancées par un périphérique), l'occurrence d'un événement (signal de réveil d'une tâche, libération d'une ressource) déclenchent l'exécution de tâches système telles que le Gestionnaire d'Interruption, l'Ordonnanceur, les Pilotes d'E/S, le "Pageur" de façon totalement aléatoire et pour un temps difficilement évaluable avec précision. La tâche s'exécutant est alors retardée par la survenance d'au minimum deux commutations de contexte (passage mode utilisateur, mode système et vice-versa), plus par le temps d'exécution de la (des) tâche(s) système.

D'autre part, l'utilisation des ressources système introduit des temps d'attente difficilement évaluables. Ces temps d'attente peuvent résulter des conflits pour l'allocation des ressources (accès disque, accès réseau), ainsi que les délais d'attente pour l'accès aux sections critiques du code.

2.1.2. Formulation du problème de l'ordonnancement

Ordonnancer les tâches d'une application temps réel doit consister à **garantir** celles-ci, c'est-à-dire à planifier l'exécution de leur requêtes[2] de façon à ce que leurs contraintes de temps soient respectées. Pour les tâches à contraintes strictes, ceci signifie que toutes leurs échéances doivent être tenues; pour les tâches à contraintes relatives, ceci amène à les ordonnancer de façon à respecter le nombre maximal de fois où l'échéance peut être dépassée ainsi qu'à chaque fois, l'amplitude autorisée du dépassement.

En face d'une surcharge et de la possibilité de fautes temporelles, il faut prendre des décisions d'ordonnancement de façon à maintenir le procédé dans un état sécuritaire en assurant l'exécution des tâches les plus importantes de l'application de contrôle.

2.2. L'approche réalisée par les théories

2.2.1. Contexte

Les ordonnanceurs développés par les théories travaillent en fait avec un modèle simplifié de tâches. Les hypothèses suivantes sont généralement faites:

- L'application ne comporte qu'une seule phase; toutes ces tâches sont toujours actives.
- Les tâches sont d'importance égale; par voie de conséquence, le critère est ignoré. En fait, il y a confusion entre urgence de la tâche et importance.
- Les tâches n'ont pas d'autres contraintes de ressources que celle constituée par l'allocation du processeur.
- Les paramètres temporels des tâches sont constants tout au long de la vie de l'application.

2.2.2. Formulation du problème théorique de l'ordonnancement

Les approches théoriques ne respectent pas en fait le vrai problème de l'ordonnancement en temps réel, car elles considèrent arbitrairement que les tâches périodiques sont plus cruciales que les tâches apériodiques. Ceci se reflète en partie par le fait que les tâches périodiques sont souvent à échéances strictes et les tâches apériodiques à échéances relatives.

Avec cette hypothèse de suprématie des tâches périodiques, le problème de l'ordonnancement est altéré et il devient:

- 1. Garantir les échéances des tâches périodiques,
- 2. Ordonnancer les tâches apériodiques sans mettre en péril les requêtes périodiques.

Ainsi, des tâches apériodiques à contraintes strictes peuvent ne pas respecter leurs échéances. Le but de l'ordonnancement n'est plus de garantir toutes les tâches à échéance stricte et en présence d'une surcharge les plus importantes d'entre elles, mais seulement de maximiser le Ratio de Garantie, c'est-à-dire le rapport

$$\frac{\text{Nombre de tâches apériodiques strictes garanties}}{\text{Nombre de tâches apériodiques strictes soumises}}$$

En exemple, nous pouvons citer les algorithmes d'acceptation de Chetto [7].

Pour les tâches apériodiques à contraintes relatives, il s'agit de leur offrir le meilleur temps de réponse possible, ce qui revient à les traiter comme des tâches de fond. Citons, pour illustration, le traitement d'arrière plan ou les méthodes à base de serveurs [5].

Il est évident que cette suprématie accordée aux tâches périodiques n'a aucun fondement sérieux. Les tâches périodiques assurent ordinairement la supervisation "courante" du procédé tandis que les tâches apériodiques sont dédiées au traitement des erreurs, des alarmes, plus généralement de toutes les situations d'exception signalant l'occurrence d'événements anormaux au sein du procédé. Il semble donc naturel de les

2 La requête d'une tâche désigne une nouvelle occurrence de la tâche.

ordonnancer en priorité de façon à maintenir le système dans un état sécuritaire.

Ce choix des théories est en partie dû au fait *que l'on sait faire* pour les tâches périodiques et *que l'on ne sait pas bien faire* pour les requêtes apériodiques.

En effet, pour les premières, les paramètres des tâches et notamment les dates d'occurrence, sont connus à l'initialisation du système et il est alors possible de vérifier "hors ligne" que la configuration formée par les tâches périodiques est valide. En effet, une caractéristique fondamentale de la séquence produite par un algorithme préemptif sur une configuration de tâches périodiques est sa cyclicité [8]. Soit S la plus tardive date de premier réveil des tâches périodiques et P le PPCM de leur période. Le processeur fait exactement la même chose au temps $t (t \geq S)$ et au temps $t = t + kP$, où k est un entier positif. Il suffit donc d'étudier la configuration sur l'intervalle $[S, S+P]$ et de la dupliquer sur les intervalles suivants. L'étude de l'ordonnançabilité d'une configuration à départ simultané se fait donc sur l'intervalle $[0,P]$; pour une configuration à départ échelonné, il faut tester l'ordonnançabilité sur $[0,S+2P]$.

Il en est autrement des tâches apériodiques qui surviennent sur un site de façon asynchrone: on ignore donc leurs dates d'occurrence et on ne peut alors vérifier leur ordonnançabilité qu'au moment où elles surviennent, donc "en ligne", ce qui se révèle être en contradiction complète avec le besoin de déterminisme et de validation des applications.

Avant d'exposer les conséquences de l'écart existant entre le véritable problème de l'ordonnancement temps réel et l'appréhension qui en est faite dans les théories, nous nous proposons d'effectuer un rapide panorama des algorithmes d'ordonnancement théoriques existants.

2.3. Les algorithmes d'ordonnancement

Nous allons successivement présenter les algorithmes les plus courants d'ordonnancement des tâches périodiques et des tâches apériodiques. En ce qui concerne les tâches périodiques, nous ne nous préoccuperons que des tâches à contraintes strictes; par contre, pour les tâches apériodiques, nous étudierons les routines d'acceptation pour les deux types de contraintes.

Définitions:

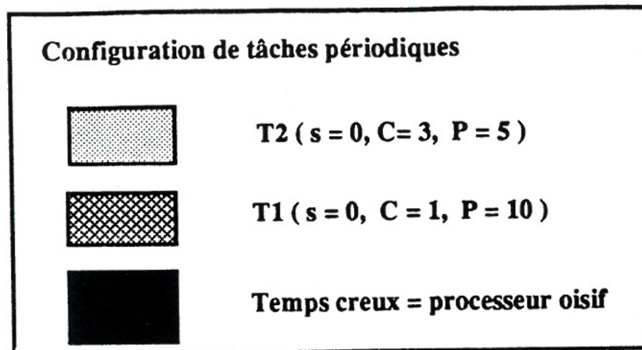
- Un ensemble de tâches est dit **ordonnançable**, **acceptable**, si toutes les requêtes de l'ensemble s'ordonnancent toujours dans le respect de leur contrainte de temps. Un **test d'acceptabilité** ou **d'ordonnançabilité** sur les paramètres temporels des tâches permet pour chaque algorithme d'ordonnancement de déterminer si un ensemble donné de tâches est acceptable avec cet algorithme.
- Un algorithme d'ordonnancement est dit **optimal**, si étant donné un ensemble de tâches acceptable, le test d'acceptabilité associé à l'algorithme décrète l'ensemble de tâches acceptable.

2.3.1. Ordonnancement des tâches périodiques

Les politiques présentées sont des politiques préemptives[3] à priorité statique ou dynamique; chacune d'elles présente un homologue non-préemptif[3] dont les performances sont différentes. Pour les premiers, la prise en compte d'une tâche se fait au réveil et à la terminaison des tâches, alors que pour les seconds, la prise en compte n'est effectuée qu'à la terminaison des tâches.

Les priorités sont soit déterminées arbitrairement (priorité empirique), ce qui présente le défaut d'ignorer les contraintes temporelles, soit calculées en fonction de l'un des paramètres de la tâche, ce qui présente l'inconvénient de devoir

3 Rappel: Un algorithme est dit non-préemptif lorsqu'il interdit l'interruption d'une tâche en cours d'exécution.Lorsqu'au contraire l'algorithme est préemptif, chaque tâche peut voir son exécution fractionnée en autant de morceaux que necéssaire.



Priorité (T2) = 1/5 > Priorité (T1) = 1/10

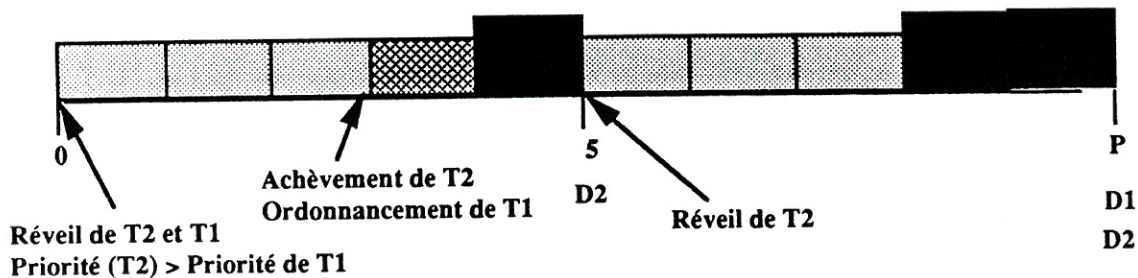
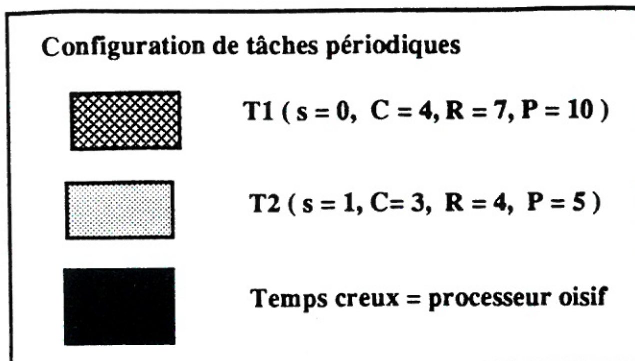


Figure 7. Illustration du Rate Monotonic



Priorité (T2) = 1/4 > Priorité (T1) = 1/7

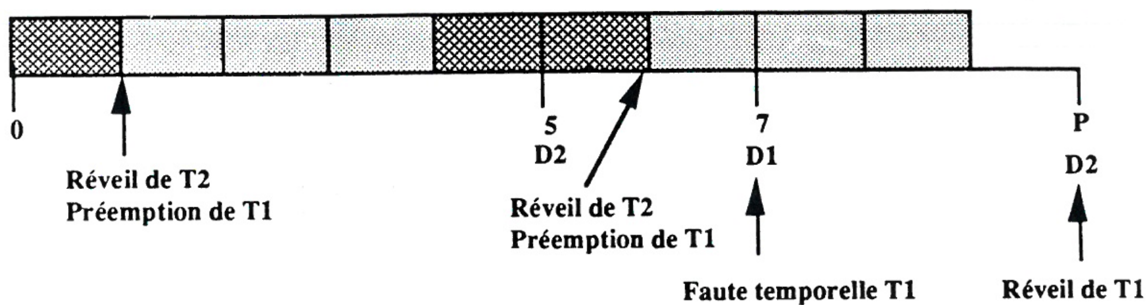


Figure 8. Illustration de Inverse Deadline

rassembler dans une seule valeur l'importance de la tâche(son poids dans l'application) et ses contraintes de temps.

2.3.1.1.Priorité empirique

La priorité de chaque tâche est déterminée par le concepteur de l'application en fonction généralement de l'importance de la tâche et de façon à régler les conflits d'accès au processeur.

La recherche des priorités et la mise au point des applications peuvent s'avérer longues et difficiles: le processus de recherche est itératif; typiquement, après avoir accordé une priorité aux tâches en fonction de leur importance, la charge du processeur est testée. Si des échéances de tâches critiques ne peuvent être respectées ou si la charge du processeur est trop faible, les priorités sont réajustées et ce jusqu'à obtenir un bon compromis.

2.3.1.2. Priorité statique

Une priorité statique est une priorité qui ne varie pas au cours de la vie de la tâche. Ce principe est simple mais il réduit les possibilités d'adaptation de la politique d'ordonnancement à l'évolution du système contrôlé. Les algorithmes les plus courants sont le Rate Monotonic [9] et l'Inverse Deadline [8].

- **Rate Monotonic**

Cet algorithme attribue une priorité fixe inversement proportionnelle à la période de la tâche: $\text{Priorité}_i = 1/P_i$. Cet algorithme est optimal pour les configurations de tâches à échéance sur requête et dans ce cas on connaît une borne minimale pour l'acceptation d'une configuration de n tâches:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{1/n} - 1)$$

Cette borne théorique correspond au pire des cas et tend vers ln 2 (69%) lorsque n est grand. Elle peut souvent être dépassée et une étude montre qu'en moyenne la borne supérieure avoisine 88%.

La Figure 7 donne un exemple de fonctionnement de l'algorithme.

- **Inverse Deadline**

Cet algorithme attribue une priorité fixe inversement proportionnelle au délai critique de la tâche: $\text{Priorité}_i = 1/R_i$. Les performances de cet algorithme sont équivalentes à celles du Rate Monotonic dans le cas de tâches à échéance sur requête et meilleures dans le cas de configurations arbitraires. La condition d'acceptabilité est la même que celle du Rate Monotonic.

La Figure 8 donne un exemple de fonctionnement de l'algorithme.

2.3.1.3. Priorité dynamique

Une priorité dynamique évolue dans le temps. Les politiques l'utilisant demandent une gestion plus importante car, à chaque fois qu'une nouvelle tâche est prête à être ordonnancée, il faut réorganiser la file d'attente du distributeur. Cependant, elles sont beaucoup plus souples et plus performantes que les précédentes. Les deux algorithmes les plus courants sont Earliest Deadline [9] [7] et Least Laxity [10] [11].

- **Earliest Deadline**

La priorité maximale est accordée à la tâche dont le délai critique dynamique est le plus faible, i.e. à la tâche dont l'échéance est la plus proche. Cet algorithme permet de garantir un plus grand nombre de configurations que le Rate Monotonic et l'Inverse Deadline, parce qu'au lieu de prendre en compte dans la priorité de chaque tâche l'écart statique date de réveil de la tâche-échéance de la tâche (ou période si les tâches sont ER), il prend pour mesure de priorité le temps restant jusqu'à l'échéance.

La Figure 9 illustre ceci avec Inverse Deadline, Rate Monotonic se ramenant à Inverse Deadline avec des tâches ER: par l'ordonnancement Inverse Deadline, la tâche T1 commet une faute temporelle qui ne se produit pas avec Earliest Deadline.

Pour des processus à échéance sur requête, une condition nécessaire et suffisante d'ordonnancement est

Configuration de tâches périodiques

T1 (s = 0, C = 4, R = 7, P = 10)

T2 (s = 1, C= 3, R = 4, P = 5)

Temps creux = processeur oisif

INVERSE DEADLINE

Priorité (T2) = 1/4 > Priorité (T1) = 1/7

0

Réveil de T2
R = 7 pour T1
R = 4 pour T2

5
D2

Réveil de T2
R = 7 pour T1
R = 4 pour T2

7
D1
Faute temporelle T1

P
D2

Réveil de T1

EARLIEST DEADLINE

0

Réveil de T2
Priorité(T2) > Priorité (T1)
R(t) = 6 pour T1
R(t) = 4 pour T2

5
D2

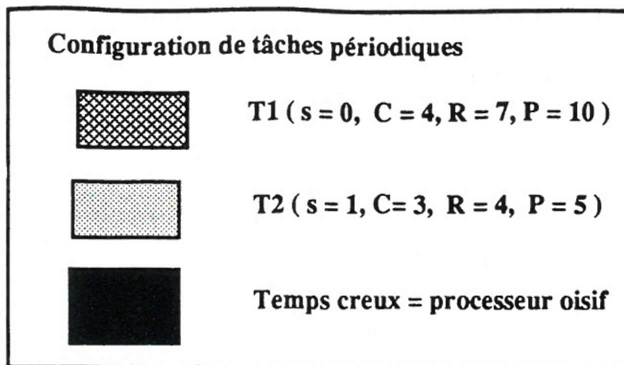
Réveil de T2
Priorité(T1) > Priorité (T2)
R(t) = 1 pour T1
R(t) = 4 pour T2

7
D1

P
D2

Réveil de T1

Figure 9. Principe de Earliest Deadline
Comparaison Earliest Deadline/Inverse Deadline



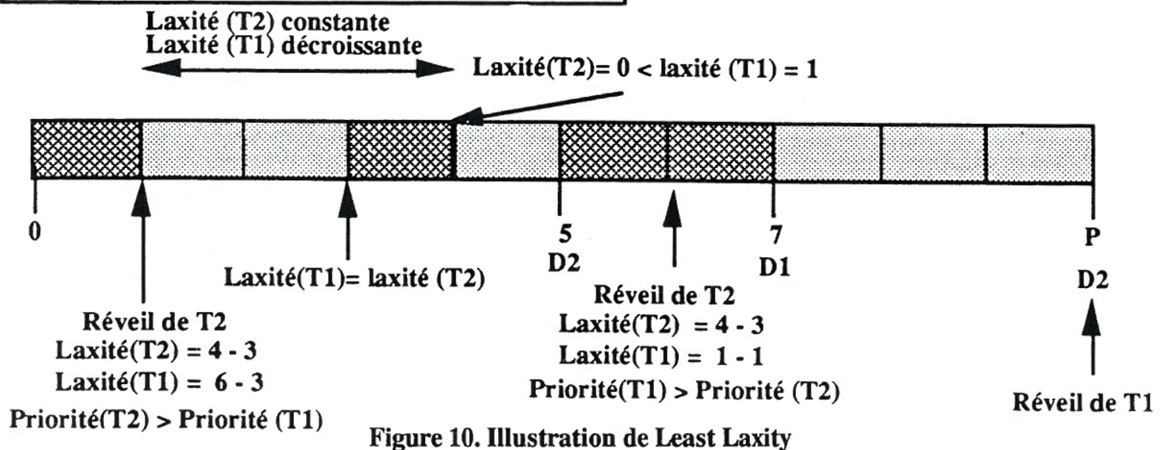


Figure 10. Illustration de Least Laxity

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Pour des tâches quelconques, une condition nécessaire et suffisante d'ordonnancement est

$$\sum_{i=1}^{n} \frac{R_i}{P_i} \leq 1$$

et

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

constitue une condition nécessaire.

- **Least Laxity**

La priorité d'une tâche est inversement proportionnelle à sa laxité dynamique $L_i(t)$, la laxité dynamique d'une tâche représentant le temps restant avant l'occurrence de sa date d'exécution au plus tard. Cet algorithme est optimal el les conditions d'acceptation d'une configuration de tâches indépendantes sans contrainte de ressources sont les mêmes que pour Earliest Deadline. Cependant, une approche comparative permet de révéler que cet algorithme présente le défaut de mener à de nombreux changements de contexte et à de nombreuses invocations de l'ordonnanceur.

La Figure 10 donne un exemple de fonctionnement de cet algorithme.

2.3.2. Ordonnancement des tâches apériodiques

On dispose d'une configuration *T* de tâches périodiques garanties. Le problème consiste donc à ordonnancer des tâches apériodiques sans remettre en cause l'ordonnançabilité des tâches périodiques de *T*. Le but recherché pour les tâches apériodiques à contraintes relatives est de minimiser leur temps de réponse, pour les tâches apériodiques à contraintes strictes, on désire maximiser le nombre de tâches acceptées.

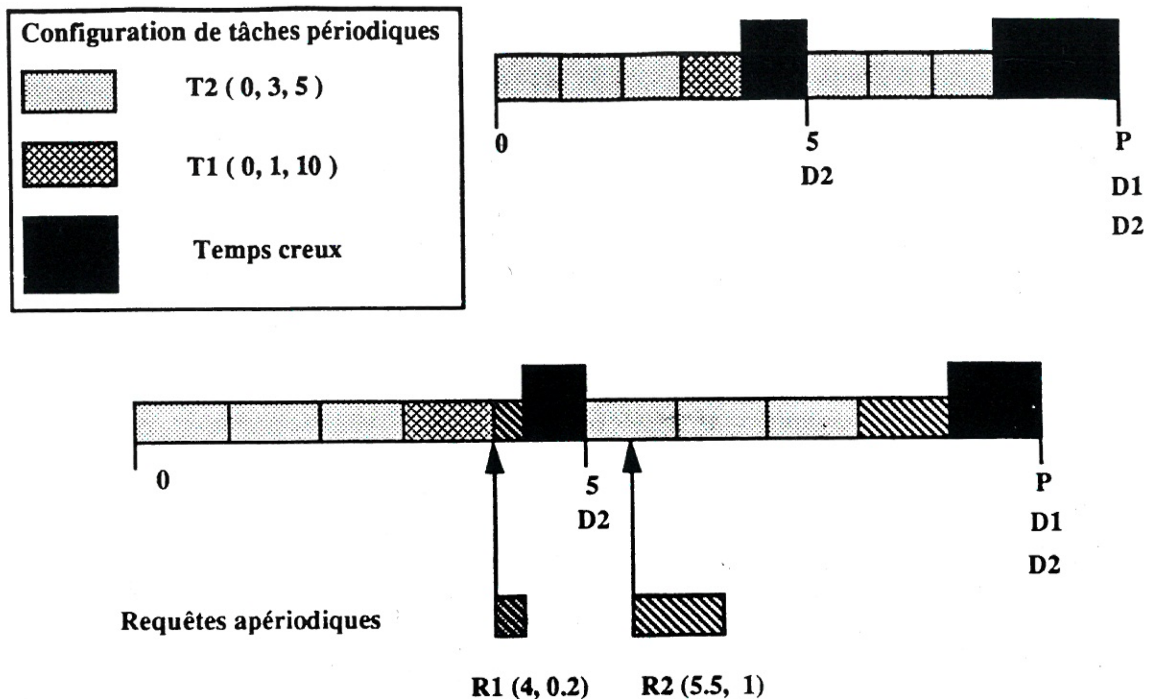


Figure 11. Traitement d'arrière plan

2.3.2.1. Tâches apériodiques à contraintes relatives

Plusieurs approches existent: la première dont nous parlerons, le **traitement d'arrière plan**, est la plus simple mais la moins performante. Nous aborderons ensuite le principe des **Serveurs de tâches** [5], puis la méthode développée par Chetto [12] [13] [14].

- Traitement d'arrière plan (background processing)

Les tâches apériodiques sont ordonnancées lorsque le processeur est oisif, i.e. lorsqu'il n'y a plus de tâches périodiques ou apériodiques acceptées précédemment à ordonnancer.

Si la charge des tâches périodiques est importante, l'ordonnanceur ne pourra pas consacrer beaucoup de temps aux tâches apériodiques et celles-ci risquent d'avoir un temps de réponse assez long.

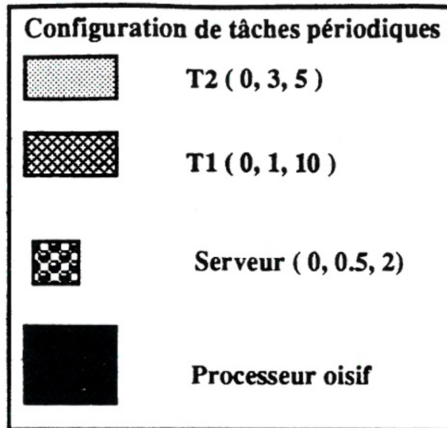
La Figure 11 présente un exemple d'application de cette méthode à une configuration de deux tâches périodiques T2 (r = 0; C = 3; P = 5) et T1 (r = 0; C = 1; P = 10). T2 reçoit par le RATE MONOTONIC la plus haute priorité.

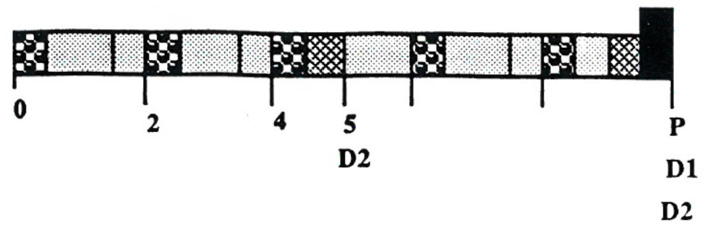
La configuration des tâches périodiques présente deux temps creux entre t = 4 et t = 5 d'une part, entre t = 8, t = 10 d'autre part. La requête apériodique R1 est immédiatement servie mais la requête apériodique R2 survenant à t = 5.5 doit attendre jusqu'à t = 8 avant de pouvoir s'exécuter. La requête R1 a un temps de réponse égal à son temps d'exécution et la requête R2 un temps de réponse égal à 3,5 fois son temps d'exécution.

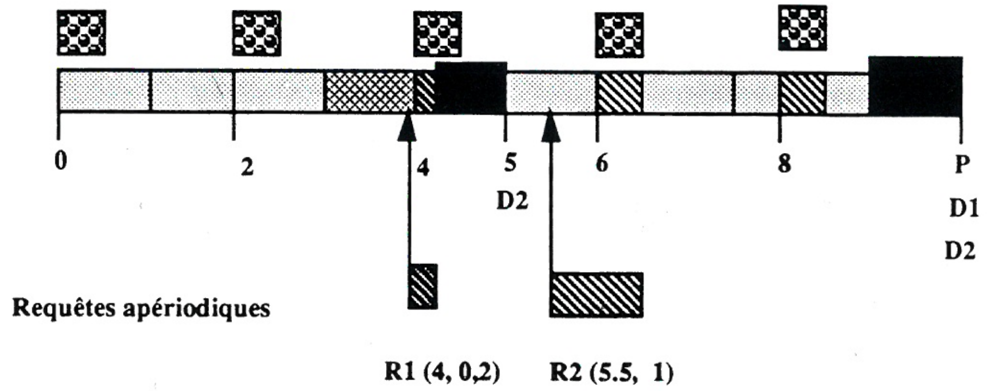
• Les serveurs de tâches

La configuration *T* est formée de tâches périodiques à échéance sur requête qui sont ordonnancées selon le Rate Monotonic.

Le SERVEUR est une tâche périodique, créée spécialement pour veiller à l'ordonnancement des tâches apériodiques. Cette tâche est généralement la tâche périodique de plus haute priorité. Plusieurs types de serveurs ont été proposés: ils diffèrent par la manière dont ils répondent aux arrivées des tâches apériodiques.







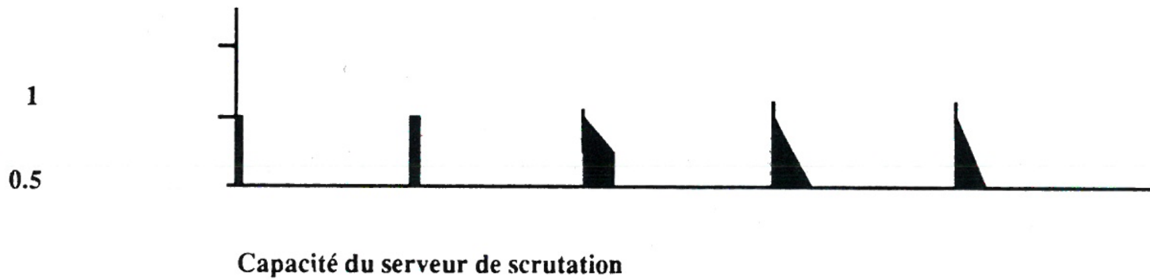
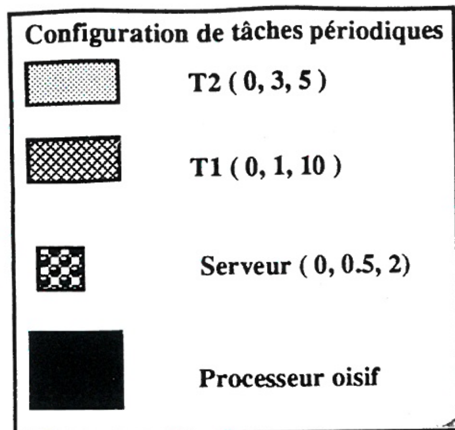
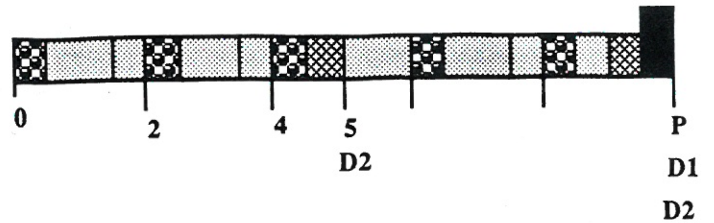
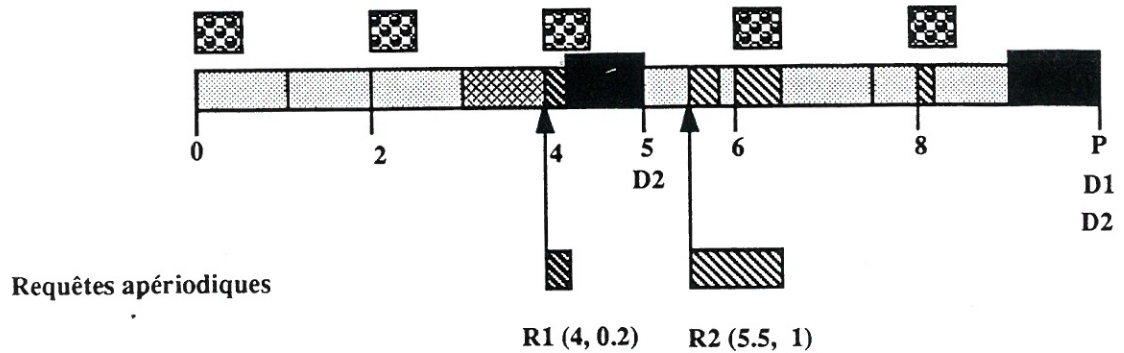
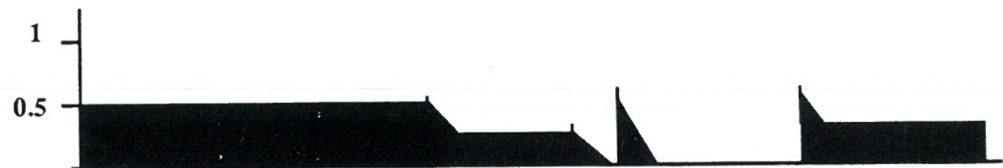


Figure 12. Serveur de Scrutation









Capacité du serveur ajournable

Figure 13. Exemple du fonctionnement du Serveur Ajournable

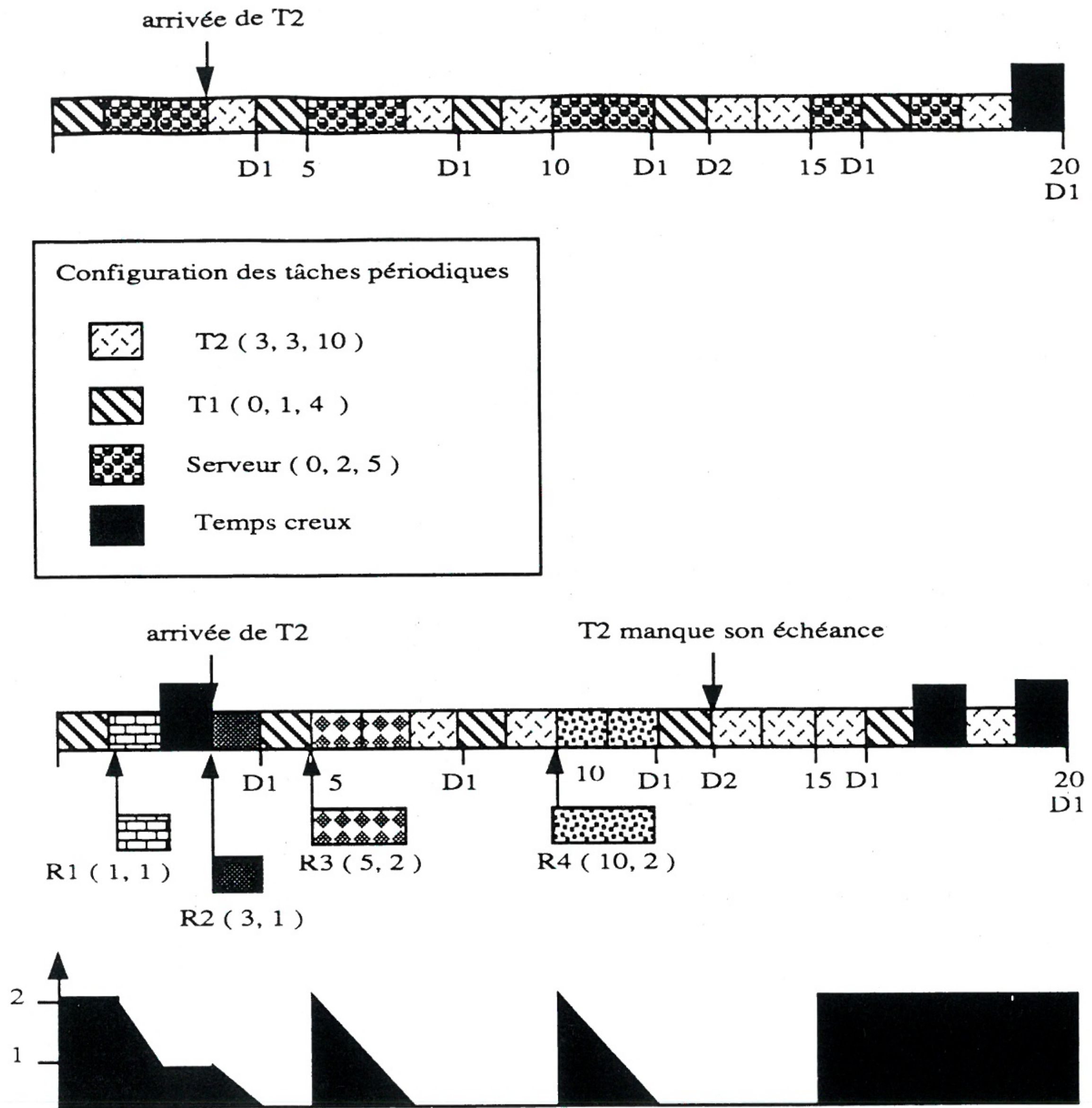


Figure 14. Exemple de non-garantie d'échéance périodique avec le Serveur Ajournable

- **Traitement par scrutation (Polling)**

Ce serveur de tâches apériodiques travaille comme suit: à chacune de ses occurrences, il traite les tâches apériodiques en suspens depuis son occurrence précédente et ce,soit jusqu'à épuisement de sa capacité (temps d'exécution), soit jusqu'à ce qu'il n'y ait plus de tâches apériodiques à servir. Si, lorsque le serveur commence son exécution, il n'y a aucune tâche apériodique en attente, le serveur se suspend et sa capacité est récupérée au profit des tâches périodiques. Cette capacité ne sera réinitialisée qu'à la prochaine occurrence du serveur.

On se rend compte facilement, qu'ici également, des tâches apériodiques peuvent avoir de grand temps de réponse, surtout si elles surviennent juste après la fin de l'exécution de la tâche serveur.

En fait, le problème principal de cette méthode est l'incompatibilité entre la nature des arrivées des tâches apériodiques (par avalanche et de manière asynchrone) et la nature périodique du serveur: d'une part, lorsque le serveur est prêt, il se peut qu'il n'y ait pas de tâches apériodiques à traiter; d'autre part, lorsque des tâches apériodiques surviennent sur le site, il se peut que le serveur ait déjà abandonné sa capacité, contraignant ainsi les tâches apériodiques à attendre sa prochaine occurrence pour être servies.

La Figure 12 montre le fonctionnement de ce serveur avec la configuration de tâches périodiques T1 et T2. La tâche serveur de scrutation s'exécute toutes les deux unités de temps; c'est la tâche de plus haute priorité. A t = 0 et à t = 2, puisqu'il n'y a pas de requête apériodique en suspens, le serveur ne s'exécute pas. La requête R1 est immédiatement servie, épuisant la capacité du serveur de 0,2 unités. Comme il n'y a aucune autre requête apériodique en attente de service, le reste de la capacité du serveur est détruite. La requête R2 ne peut être prise en compte qu'à t = 6; elle effectue la moitié de son traitement en épuisant totalement la capacité du serveur et doit encore attendre jusqu'à la nouvelle occurrence du serveur (t = 8) pour pouvoir s'achever. La requête R1 a un temps de réponse égal à son temps d' exécution; la requête R2 a un temps de réponse égal à 3, soit trois fois son temps d'exécution.

- **Le Serveur Ajournable**

Le Serveur Ajournable (Deferrable Server) découle directement du serveur utilisé pour réaliser la scrutation. Simplement, si aucune tâche apériodique n'est en suspens lors de son occurrence, il conserve entière sa capacité de traitement au lieu de la perdre au profit des tâches périodiques. De même que pour le serveur de scrutation , la capacité du serveur est remise à sa valeur initiale à chaque nouvelle occurrence de celui-ci.

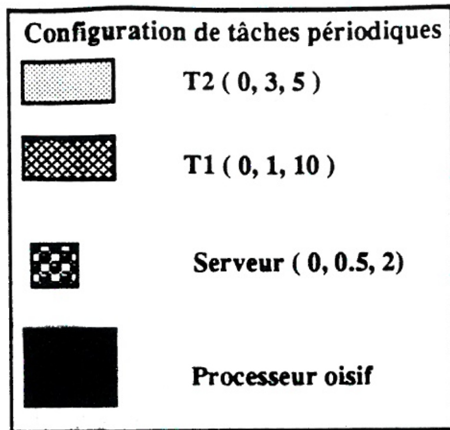
Ainsi, une tâche apériodique survenant juste après l'occurrence du serveur, peut être immédiatement servie et n'a plus à attendre l'occurrence suivante.

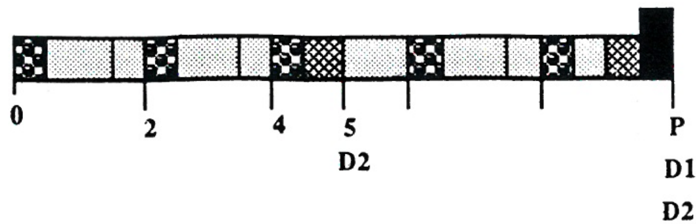
Le problème essentiel posé par le Serveur Ajournable est qu'il ne respecte pas une des règles principales de l'ordonnancement selon le Rate Monotonic des tâches périodiques, à savoir que la tâche prête de plus haute priorité doit immédiatement s'exécuter; le recul de son exécution pouvant entraîner la violation des échéances des tâches de priorité inférieure. En effet, lorsque le Serveur est élu pour s'exécuter et qu'il n'y a pas de tâche apériodique en suspens, il réserve sa capacité et abandonne son tour aux tâches périodiques prêtes de priorité inférieure à la sienne et s'exécute plus tard, dès que survient une requête apériodique.

La Figure 13 présente un exemple de fonctionnement du Serveur Ajournable pour la configuration périodique T1 et T2. A t = 0 et t = 2, il n'y a pas de requêtes apériodiques à servir; le serveur ne s'exécute pas mais garde sa capacité. A t = 4, la capacité du serveur est utilisée pour immédiatement servir la requête R1 et s'épuise donc de 0,2 unités. A t = 5,5, la requête R2 est servie en utilisant la capacité restante du serveur; son exécution reprend à t = 6 puis à t = 8 lorsque la capacité est réinitialisée. La requête R1 a un temps de réponse égal à son temps d'exécution; la requête R2 a un temps de réponse égal à 2,7 soit un peu plus de deux fois son temps d'exécution.

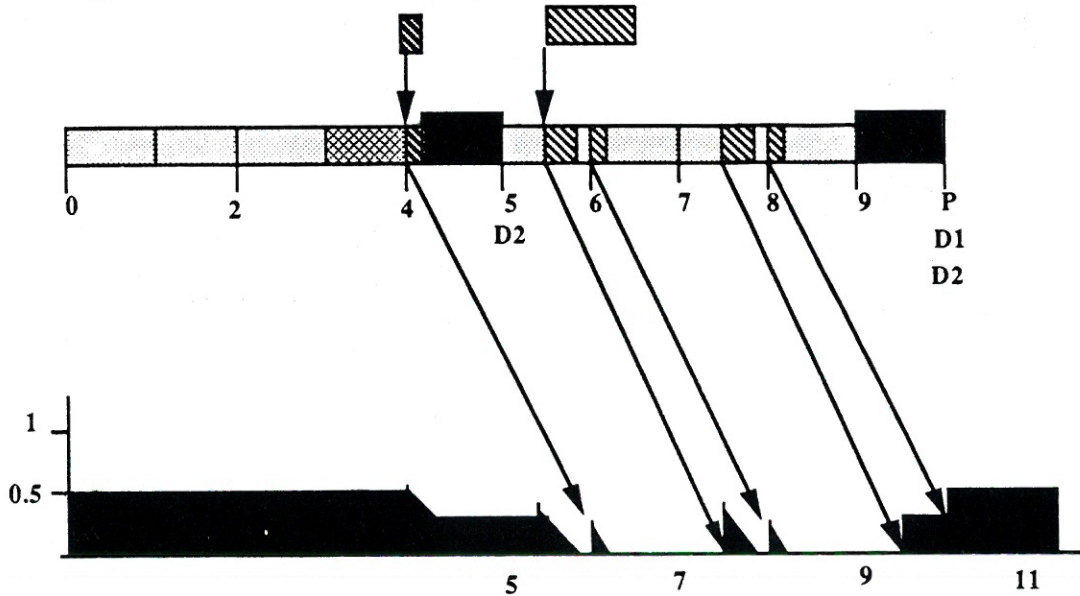
Dans cet exemple, les échéances des tâches périodiques sont respectées.

La Figure 14 présente, au contraire, un cas où l'exécution du Serveur Ajournable conduit à dépasser l'échéance d'une des tâches périodiques.





Requêtes apériodiques



Capacité du serveur sporadique

Figure 15. Exemple de fonctionnnement du serveur sporadique

La configuration périodique est formée de deux tâches T1 (0,1,4) et T2 (3,3,10) et du serveur qui a une capacité de deux unités de temps. Le premier diagramme en haut de la Figure montre que la configuration est ordonnançable lorsque le Serveur Ajournable s'exécute dans les intervalles de temps qui lui sont réservés. Voyons ce qui se passe lorsque des requêtes apériodiques surviennent: la tâche T1, de plus haute priorité, est exécutée dès qu'elle se réveille et ainsi parvient à respecter son échéance. Par contre, la tâche T2, de plus petite priorité, est gênée dans son déroulement par le service des requêtes apériodiques effectué à un niveau de priorité plus élevé que le sien et de ce fait, dépasse son échéance: la requête R2 survenant juste au moment où T2 se réveille empêche la tâche T2 de s'exécuter entre les temps t = 3 et t = 4 comme elle le faisait dans la configuration périodique; le serveur étant ensuite actif à chacune de ses occurrences ultérieures, cette unité de temps perdue ne peut être rattrapée.

- **Le Serveur Sporadique**

De façon similaire au Serveur Ajournable, le Serveur Sporadique est une tâche périodique de haute priorité qui conserve sa capacité de traitement. Il diffère cependant dans la façon dont il réinitialise celle-ci: au lieu de le faire périodiquement (à chacune de ses occurrences), il le fait à chaque fois que sa priorité active (i.e. **une tâche de priorité supérieure s'exécute ou le serveur s'exécute lui- même**), et que sa capacité n'est pas nulle. DATE_INIT, la date de prochaine réinitialisation, prend pour valeur la date courante additionnée de la période du serveur. La valeur de réinitialisation VAL_INIT est égale à la totalité de la capacité du serveur consommée.

Cette méthode permet de toujours garantir le respect des échéances des tâches périodiques. En effet, lorsque le Serveur Sporadique diffère son exécution d'une façon similaire à celle du Serveur Ajournable, il diffère également la date de réinitialisation de sa capacité d'une durée égale à sa période, lorsque celle-ci est épuisée, laissant ainsi du temps disponible pour l'exécution des tâches périodiques de priorité moindre.

La Figure 15 présente un exemple d'utilisation du Serveur Sporadique, toujours avec la configuration de tâches périodiques T1 et T2 des Figures 1,2 et 3. A t = 4, la requête apériodique R1 absorbe 0,2 unités de la capacité du serveur. DATE_INIT est positionnée à t+P = 6 et VAL_INIT est équivalente à la capacité consommée soit 0,2. R2 survenant à 5.5 épuise la capacité restante du serveur. Son exécution reprend ultérieurement à chaque fois que la capacité du serveur est réinitialisée soit à t = 6, t = 5.5+2, t = 6+2 et consomme toujours toute la capacité accordée (VAL_INIT successivement égale à 0.2, 0.3, 0.2). A t = 9.5 (7.5+2), le serveur reprend une capacité égale à 0.3 puis, à t = 10 (8+2) retrouve sa pleine capacité.

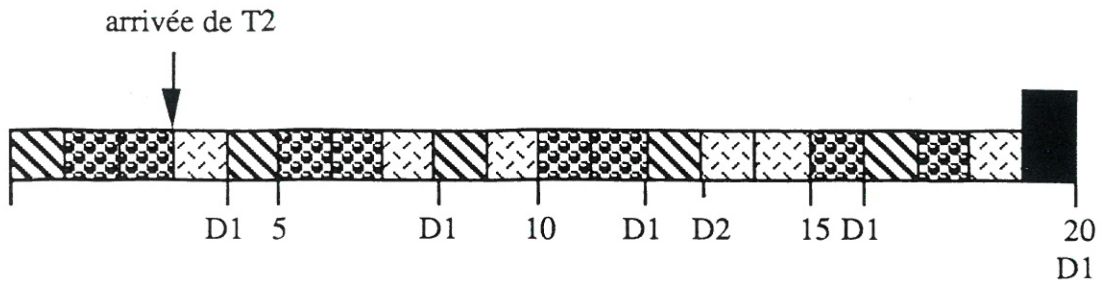
La requête R1 a un temps de réponse égal à son temps d'exécution, tandis que la requête R2 a un temps de réponse égal à 2.7.

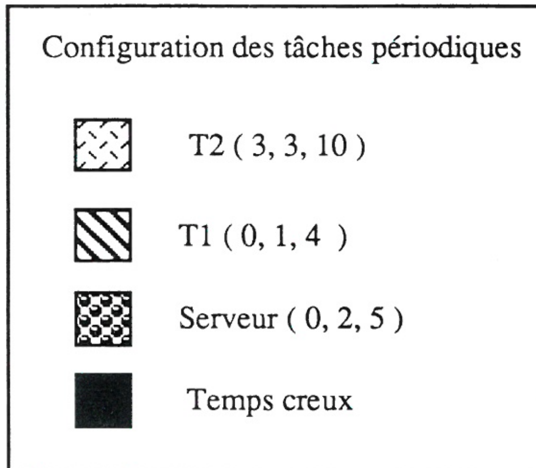
La Figure 16 reprend la configuration de la Figure 12 et permet de montrer que cette fois, par l'utilisation du Serveur Sporadique, les échéances des deux tâches périodiques T1 et T2 sont respectées. A t = 0, T1 s'exécute et sa priorité étant supérieure à celle du serveur, celui-ci devient actif: DATE_INIT est donc réinitialisée et prend pour valeur 0 + 5 = 5. Lorsque R1 est exécutée, VAL_INIT est fixé à 1. Comme dans le cas du Serveur Ajournable, à t = 3, la requête R2 survenant au réveil de la tâche T2 empêche celle-ci de s'exécuter et épuise la capacité du serveur (on a alors un nouvel DATE_INIT = 8 et VAL_INIT = 1). Cependant, par la suite, à cause de la manière dont il réinitialise sa capacité, le Serveur Sporadique ne peut pas complètement servir les requêtes apériodiques R3 et R4 comme le faisait le Serveur Ajournable et la tâche T2 peut rattraper l'unité de temps perdue entre t = 3 et t = 4 entre les instants 6 et 7.

- **La méthode de Chetto**

La configuration T est formée de tâches périodiques quelconques ordonnancées selon l'Earliest Deadline.

L'idée de cette méthode est de transformer les tâches apériodiques à contraintes relatives en tâches apériodiques à contraintes strictes par l'ajout à leurs paramètres temporels d'une échéance fictive. Ainsi, les tâches apériodiques pourront être ordonnancées grâce à un algorithme traitant le cas des tâches apériodiques à contraintes strictes, notamment avec l'algorithme conjoint Earliest Deadline que nous présentons au





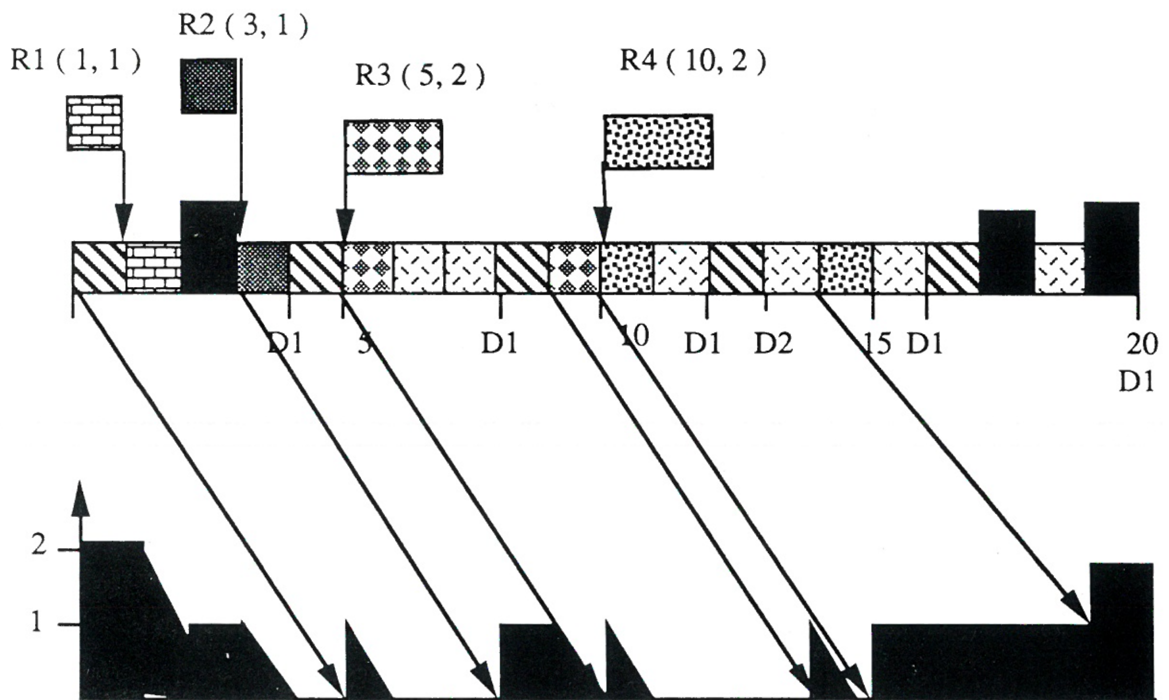


Figure 16

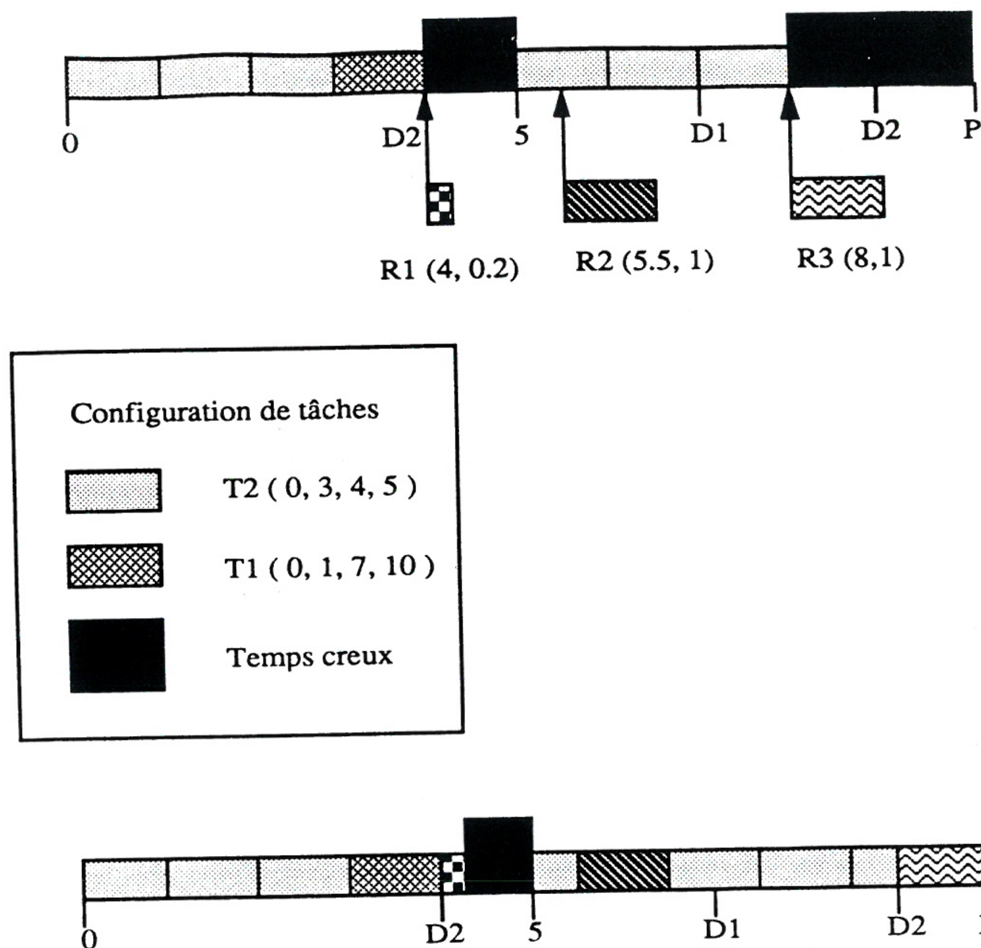


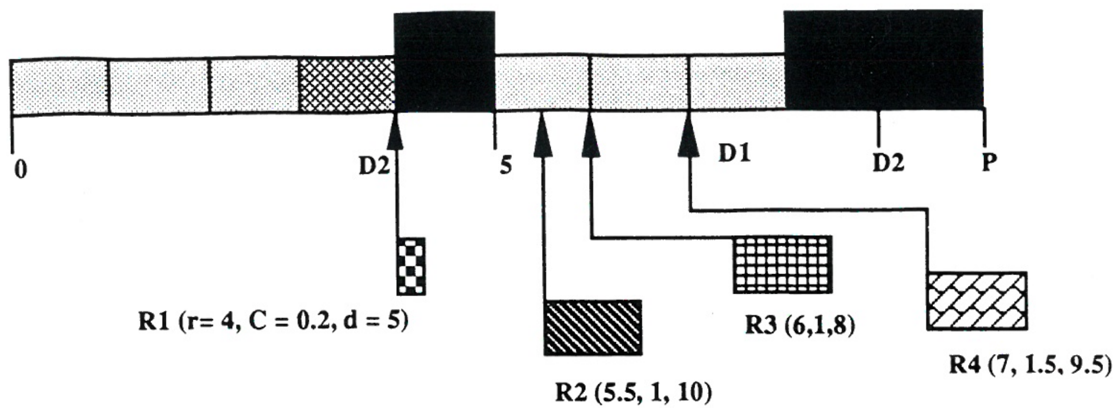
Figure 17. Exemple du mécanisme à échéance fictive

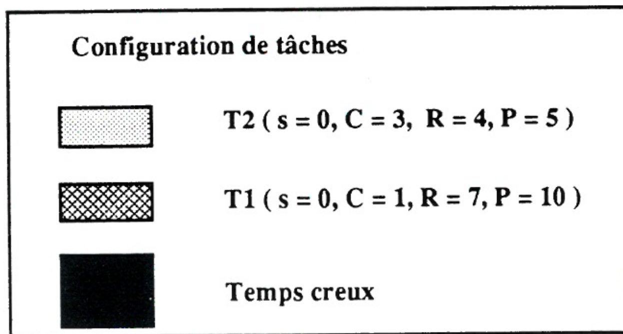
paragraphe traitant des tâches apériodiques à contraintes strictes.

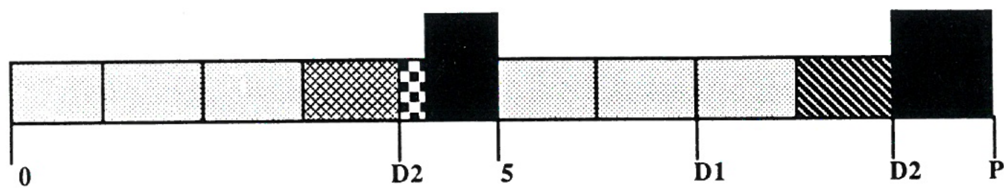
A chaque fois qu'une tâche apériodique survient,il faut donc calculer son échéance fictive Df telle qu'elle soit l'échéance minimale qu'il faut associer à la tâche, pour que celle-ci s'exécute avec un temps de réponse minimal. Df correspond alors à la première date pour laquelle le temps total d'inactivité du processeur ordonnançant dans le respect de leurs échéances les tâches périodiques et les tâches apériodiques précédemment déclenchées et non encore achevées, est égal au temps d'exécution de la tâche.

Concrètement, les tâches apériodiques sont gérées à l'ancienneté; cependant, cette méthode présente plusieurs avantages par rapport à un simple ordonnancement basé sur une file FIFO et une utilisation des temps creux:

- un avantage d'optimalité: par le calcul des échéances fictives, les tâches apériodiques sont mises sur le même plan que les tâches périodiques et leur temps de réponse est alors minimal.
- un avantage de simplification par l'utilisation d'un même ordonnanceur EDS pour les tâches périodiques et les tâches apériodiques (pas de gestion spéciale







La tâche R1 dipose d'un temps creux suffisant entre [4, 5]

La tâche R2 dispose d'un temps creux suffisant entre [8,10]

La tâche R3 ne peut pas être garantie car il n'existe pas assez de temps creux

La tâche R4 ne peut être garantie car sinon R2 ne peut rester garantie: il existe un temps creux suffisant entre [8, 9,5]. Seulement si R4 est acceptée, elles s'ordonnance selon Earliest Deadline avant R2, donc entre t = 8 et 9,5. et R2 qui s'ordonnance juste après commet une faute temporelle. En conséquence R4 est rejetée.

Figure 18. Ordonnancement dans les temps creux d'une séquence rigide de tâches périodiques Earliest Deadline

contrairement aux serveurs).

La Figure 17 donne un exemple d'ordonnancement de deux requêtes apériodiques R1 (4,0.2) et R2 (5.5,1) dans la configuration EDS des tâches périodiques T1 et T2. A t = 4, date d'occurrence de R1, on dispose du temps compris entre t=4 et t=6 pour l'ordonnancer , la seconde requête de T2 restant garantie (au plus tard, elle doit démarrer son exécution à t = 6). R1 peut immédiatement être ordonnancée; son temps d'exécution étant de 0.2 unité, Df = 4 + 0.2 = 4.2. La tâche critique R'1 correspondant à R1 est donc R'1 (4, 0.2, 4.2) et elle offre bien un temps de réponse minimal à R1. Lorsque la tâche R2 survient, la requête T2 a déjà travaillé pendant 0,5 unité; au plus tard pour s'achever dans le respect de son échéance, elle doit reprendre son exécution à t = 6.5. On dispose donc d'un temps creux [5.5, 6.5] dans lequel R2 peut être insérée. R'2 est donc (5.5, 1, 6.5). La requête R3 survenant à t = 8 ne peut commencer son exécution qu'après que la requête de la tâche T2 soit achevée: en effet, cette dernière a une laxité nulle et doit poursuivre son exécution. R3 ne pourra donc être prise en compte qu'après, à t = 9. Son échéance fictive est en conséquence fixée à t = 10. On a R'3 (8, 1, 10).

2.3.2.2. Tâches apériodiques à contraintes strictes

Il existe deux méthodes: la première consiste à rapprocher le modèle des tâches apériodiques de celui des tâches périodiques en dotant celles-ci d'une pseudo-période, représentant l'intervalle minimal entre deux occurrences successives d'une tâche apériodique [5]. La seconde traite les tâches apériodiques sans faire aucune supposition sur le rythme de leurs arrivées: à chaque occurrence de tâches apériodiques, un algorithme appelé "Routine de Garantie" teste si la nouvelle tâche pourra être garantie sans mettre en péril les tâches périodiques et les tâches apériodiques précédemment acceptées. Si non elle est rejetée.

- **Première approche: Introduction d'une pseudo période Π**

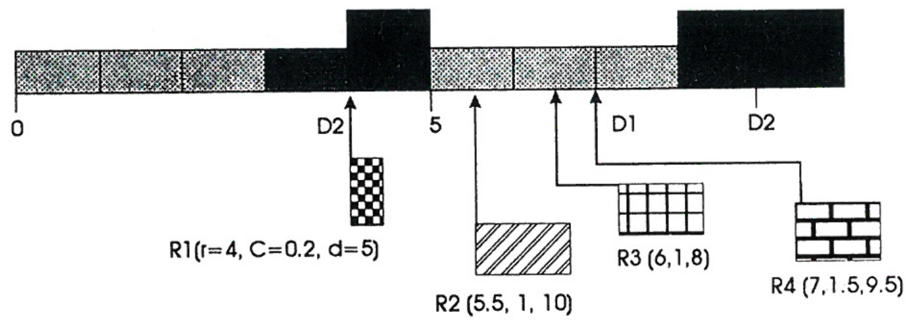
Le modèle des tâches apériodiques est rapproché de celui des tâches périodiques par le biais d'une pseudo-période Π qui représente l'intervalle de temps minimal séparant deux occurrences successives de la tâche. Une tâche apériodique R_i est alors représentée par le quadruplet (r_i, C_i, d_i, Π_j) et, dans le pire des cas, elle arrive sur un site à toutes les dates $r_i + k\Pi_j$. Sur le plan théorique, on a donc éliminé tout caractère aléatoire et on ne travaille plus qu'avec un seul modèle de tâches celui des tâches périodiques, modèle que l'on sait correctement (ou presque) traiter. Pratiquement, les tâches apériodiques sont mises au même plan (dans le sens de l'importance) que les tâches périodiques.

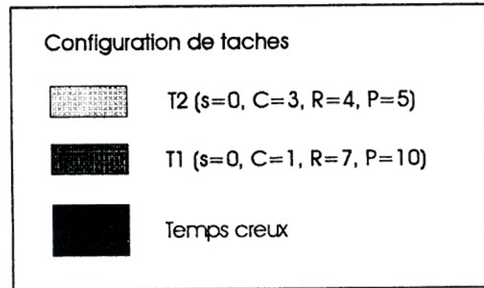
Le principal défaut de ces méthodes est constitué par cette obligation de calquer le modèle des tâches apériodiques sur celui des tâches périodiques. La pseudo-période introduite pour cela semble quelque chose de difficilement évaluable, voire même de tout à fait irréaliste. La plupart des tâches apériodiques strictes d'un système temps réel étant dédiées au traitement de conditions d'exception survenant dans le système (alarme), elles sont susceptibles, si tout va bien, de ne jamais s'exécuter et en tous les cas très rarement et de façon tout à fait aléatoire. Quelle période leur donner? Rien ne permet à l'initialisation du système de savoir quand ces tâches vont s'exécuter, ni de quel intervalle de temps, deux hypothétiques alarmes seront séparées. De plus, cette méthode infère automatiquement une sous-utilisation du processeur puisque la configuration initiale, bâtie et testée en prenant compte de l'exécution "périodique" des tâches apériodiques, va rarement se dérouler dans son entier.

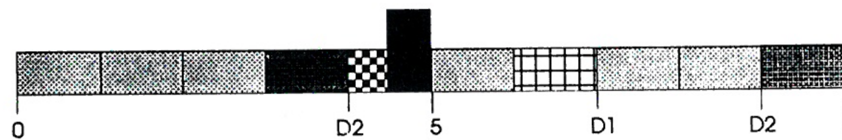
- **Deuxième approche: Routine de Garantie**

La deuxième approche ne fait aucune supposition sur la fréquence d'arrivée des tâches; celles-ci peuvent donc survenir à tout instant.

Chetto [7] présente plusieurs algorithmes dynamiques d'acceptation de tâches apériodiques, pour d'une part des configurations de tâches périodiques à départ simultané et d'autre part, pour des configurations de tâches périodiques à départ échelonné. Ces algorithmes diffèrent par ailleurs, selon un critère d'optimalité et d'importance accordée aux tâches apériodiques.







Réveil de R1: l'ensemble des tâches connues du systéme est composé des requêtes R1 et T2.. Selon Earliest Deadline, ces deux tâches s'ordonnancent dans le respect de leurs échéances; R1 est donc acceptée. R1 est la tâches de plus proche échéance, elle est élue et s'exécute donc à partie de t=4.

Réveil de R2: l'ensemble des des tâches connues du système est composé des requêtes R2 et T2. (Crestant=2,5). Selon Earliest Deadline, ces deux tâches s'ordonnancent dans le respect de leurs échéances; R2 est donc acceptée. T2 est la tâche de plus proche échéance, elle est élue et reprend donc son exécusion.

Réveil de R3: l'esemble des tâches connues du système est composé de requêtes R2, R3 et T2. (Crestant=2). Selon Earliest Deadline, ces deux tâches s'ordonnancent dans le respect de leurs échéances; R3 est donc acceptée. R3 est la tâche de plus proche échéance, elle est élue et commence son exécution à t=6.

Réveil de R4: l'esemble des tâches connues du système est composé de requêtes R2, R3 et T2. (Crestant=1). Selon Earliest Deadline, ces deux tâches s'ordonnancent dans le respect de leurs échéances; la requête R2 commet une faute temporelle. En conséquence, la tâche R4 ne peut être garantie et est rejetée.

Figure 19. Ordonnancement conjoint

- **Ordonnancement dans les temps creux d'une séquence rigide de tâches périodiques ED**:
  La routine de garantie teste si la nouvelle requête apériodique peut être placée dans les temps creux de la configuration périodique ordonnancée selon Earliest Deadline et ce dans le respect de toutes les échéances.
  La configuration des tâches périodiques est ordonnançable et reste immuable. Elle présente des temps creux, i.e. des instants où le processeur est oisif. On y place l'ensemble des requêtes apériodiques en l'ordonnançant selon Earliest Deadline.

Lorsqu'une nouvelle requête apériodique survient, la question est:

1/ Existe-t-il entre la date de réveil et l'échéance de la nouvelle requête assez de temps creux afin que l'échéance de la requête soit respectée?

2/ Si oui, et la nouvelle requête étant prise en compte, est-ce que la nouvelle acceptation ne met pas en péril les tâches apériodiques précédemment acceptées?

La Figure 18 illustre le fonctionnement de l'algorithme.

- **Ordonnancement conjoint avec recherche d'une nouvelle séquence pour les deux types de tâches:** (utilisé pour le mécanisme des échéances fictives):
  A chaque arrivée d'une nouvelle tâche apériodique, il y a construction d'une nouvelle séquence Earliest Deadline comprenant à la fois les requêtes périodiques, les tâches apériodiques précédemment acceptées et non achevées et la nouvelle tâche.
  La Figure 19 illustre le fonctionnement de l'algorithme.

La deuxième routine de garantie est contrairement à la première optimale et présente également l'avantage d'ordonnancer au même niveau les requêtes périodiques et les tâches apériodiques acceptées. Cependant, il est plus coûteux en temps de calcul.

3. Les conséquences de l'écart existant entre le véritable problème de l'ordonnancement et le problème théorique.

Toutes les hypothèses faites par les approches théoriques et la suprématie accordée aux tâches périodiques font que les théories sont distantes des vrais problèmes et sont difficilement utilisables pour résoudre ceux-ci.

Aussi, pour les concepteurs d'applications temps réel, appliquer les algorithmes théoriques d'ordonnancement a deux conséquences immédiates:

- d'une part, ils doivent adapter leurs applications au modèle théorique: en premier lieu, donc, ils doivent considérer que celles-ci ne comportent qu'une phase de fonctionnement et ensuite ils doivent fixer les paramètres temporels de leurs tâches - période et temps d'exécution,
- d'autre part, ils ne disposent d'aucune certitude sur le comportement de leur applications en face d'une surcharge car les théories prennent notamment mal en compte les requêtes apériodiques.

3.1. Adaptation des applications au modèle théorique

Toutes les tâches de l'application doivent être considérées comme étant actives et leurs paramètres doivent être fixés de façon à assurer le respect des échéances strictes, donc en choisissant les pires valeurs possibles. Ainsi, pour chaque tâche, il faut travailler avec:

- sa plus petite période. Or, cette fréquence maximale d'activation correspond généralement à des phases particulières, ponctuelles de la vie de l'application, phases durant lesquelles le suivi de l'environnement doit être plus poussé.
- son plus grand temps d'exécution possible, temps qui peut se révéler être largement supérieur au temps d'exécution moyen de la tâche.
  Plusieurs études se préoccupent d'une méthode pour déterminer le pire temps d'exécution. Nous exposons ci-dessous en particulier celle de (LEINBAUGH 80) afin

de mettre en évidence leur difficulté d'application.

Détermination du pire temps d'exécution d'une tâche

Leinbaugh dans [15] analyse les éléments influant les temps d'exécution, en tentant de prendre en compte les temps d'attente pour les ressources et les surcoûts engendrés par l'exécution du système. L'exécution d'une tâche est divisée en deux types de segments, les uns utilisant des ressources système, les autres non. Partant de la constatation que le surcoût engendré par l'emploi d'une ressource ou d'un périphérique se situe juste avant et juste après l'accès, celui-ci est intégré dans le temps d'utilisation de la ressource. Seules les interruptions horloge posent un problème car on ne peut déterminer à l'avance leur place dans les segments d'exécution. Leur prise en compte est réalisée en comptant le nombre de fois où elles se déclenchent sur le temps total de l'exécution de la tâche.

La formule suivante du pire temps d'exécution est donnée:

Pire_tps_exec = TOTR + TOTD + +TOTN/Rate + B + TO

où

- TOTR est le temps d'exécution pour les segments à ressource, prenant en compte les surcoûts d'accès.
- TOTD est le temps passé pour les opérations sur des périphériques, prenant en compte les temps d'accès et de reconnaissance et traitement de l'IT de fin de travail.
- TOTN/Rate est le temps d'exécution pour les segments sans ressource.
- B est le temps durant lequel la tâche est ralentie ou bloquée par d'autres tâches.
- TO est le temps maximal passé à traiter des ITs, durant l'exécution de la tâche.

Cette approche nécessite évidemment de connaître de façon précise les surcoûts pour l'accès aux ressources, les temps d'exécution des appels système ainsi que le temps consommé par le traitement des interruptions. L'une des clefs de voûte de cette méthode est donc de posséder des mesures fiables du temps d'exécution du système, i.e. de rendre le comportement de la machine prévisible. Notamment, il faut optimiser les temps de commutation, la prise en compte des interruptions et ôter de la gestion du système tous les mécanismes pouvant être source de délais aléatoires comme ceux de la mémoire virtuelle (pagination) et des caches. De nombreux systèmes temps réel ainsi annoncent des temps de commutation de contexte et de temps de réponse à une interruption (par exemple, DUNE-IX[4] resp 25 μs et 300 μs; VRTX[5] resp 10 μs et 15 μs; RTC[6] resp 15 μs et 8 μs [16]); cependant, d'autres facteurs influençant tout autant les performances comme le temps maximum de masquage des interruptions dans l'exécution des primitives (temps de latence) ou le temps de réarmement d'une interruption sont rarement communiqués car généralement difficilement évaluables ou dépendant des circuits périphériques utilisés.

Au niveau des temps d'accès aux ressources, des protocoles spécifiques ont pour objectif de réduire ces derniers ou d'améliorer leur caractère déterministe, notamment en ce qui concerne les attentes sur les sections critiques et le médium de communication:

- Section Critique: il s'agit de faire face aux inversions de priorité, en réduisant au maximum le temps durant lequel une tâche est bloquée à l'entrée d'une section critique par une tâche de plus faible priorité la possédant [17].
- Médium de communication: il s'agit de garantir des délais d'accès et des temps de réponse finis, bornés et a priori connus. Des protocoles dédiés comme GAM-T-103 [18]

4 DUNE-IX de Dune Technologies
5 VRTX de ready-Systems
6 RTC de GSI-Tecsi

ou CSMA/CDR[7], VTCSMA- L, VTCSMA-D[8] [19] [20] [21] visent à réduire la complexité des couches d'accès ou le nombre de collisions à l'émission sur le réseau.

Sous ces conditions d'activité de toutes les tâches de l'application et d'utilisation des pires valeurs pour les paramètres temporels des tâches, le processeur peut être sévèrement sous-utilisé. Reprenons l'arbre de recherche de [6], pour les applications temps réel de l'intelligence artificielle: le pire cas d'exécution est largement supérieur au temps moyen. Un exemple sur un parcours d'arbre de profondeur dix et d'arité trois montre que le pire cas d'exécution est mille fois plus grand que le temps moyen. Baser l'ordonnancement des tâches sur celui-ci conduit à avoir un usage moyen du processeur inférieur à 0.001.

Par ailleurs, au cours de l'ordonnancement, de mauvaises décisions peuvent être prises, i.e. une tâche peut être considérée comme non-garantie et être abandonnée alors qu'elle pourrait réellement s'achever avant son échéance.

3.2. La difficulté de faire face aux surcharges

En utilisant les algorithmes d'ordonnancement proposés par la théorie, les concepteurs d'applications temps réel n'ont aucune assurance que le procédé sera, quoiqu'il advienne, toujours dans un état sécuritaire, car lors d'une surcharge du système, ils n'ont aucun contrôle sur l'identité des tâches non garanties et celles-ci peuvent alors être des tâches de haute importance.

Par exemple, le Rate Monotonic[9] [9] amène les tâches de plus longue période à manquer leur échéance, tandis qu'avec l'Earliest Deadline[10] [9], a priori, il paraît impossible de déterminer quelles tâches vont être mises en péril: un série de simulations a été effectuée au CNAM dans le cadre d'un mémoire d'ingénieur [22]; elles montrent qu'en situation de surcharge ou avec des tâches à temps d'exécution variables, ED opère en fait une sorte de nivellement du nombre de fautes temporelles et cherche à fournir des résultats équivalents pour toutes les tâches au lieu de favoriser les plus importantes.

Ainsi pour être certains que les tâches les plus importantes de leurs applications s'exécuteront toujours de manière à correctement piloter le procédé, les concepteurs doivent soit incorporer un mécanisme particulier de reprise sur faute temporelle, soit surdimensionner leur système pour pouvoir exécuter les tâches apériodiques de traitement d'exception.

3.2.1. Les mécanismes de reprise sur faute temporelle

Deux approches se distinguent au sein des propositions actuelles: le Mécanisme de Bipartition[11] [23][24] et la Méthode du Calcul Approché[25].

Mécanisme de Bipartition

Cette méthode consiste à implanter chaque tâche en deux versions. Une version dite **Primaire**

---

7 CSMA/CDR (Carrier Sense Multiple Access/Deterministic Collision Resolution) résout de manière arborescente les collisions survenues au cours d'une émission.

8 VTCSMA-L (Virtual Time Carrier Sense Multiple Access-Laxity) et VTCSMA-D(Virtual Time Carrier Sense Multiple Access-Deadline) ordonnent les émissions de chaque site respectivement en fonction de leur laxité ou de leur échéance.

9 Rate Monotonic (RM) est un algorithme à priorité statique qui donne la plus forte priorité à la tâche de plus petite période.

10 Earliest Deadline (ED) est un algorithme à priorité dynamique qui donne la plus forte priorité à la tâche dont l'échéance est la plus proche.

11 Cette technique introduite sous le vocable initial "Deadline Mechanism" a été traduit en français par "le mécanisme à échéance". Cependant, ce terme nous apparaissant impropre et surtout ambigu du fait de l'existence du terme "algorithme conduit par échéance", nous en avons adopté un moins proche "le mécanisme de bipartition" qui est plus approprié et qui permet d'éviter la confusion.

produit une bonne qualité de service mais au bout d'un temps indéterminé. Une version dite **Secondaire** fournit un résultat seulement acceptable mais son temps d'exécution est connu à l'initialisation. Le principe suivant est appliqué: le respect de toutes les échéances doit être assuré soit par le primaire et à défaut par le secondaire. Lorsque le primaire échoue, il faut alors assurer l'exécution du secondaire.

Mécanisme du Calcul Approché

Dans cette approche, une tâche est divisée en deux parties, une partie dite **Mandataire** et une partie dite **Optionnelle.** La partie mandataire doit obligatoirement s'exécuter dans le respect de son échéance; elle fournit un résultat approché; la partie optionnelle, affinant le résultat, est exécutée s'il reste assez de temps.

Ces méthodes ont l'inconvénient de ne pas pouvoir s'appliquer à toutes les tâches d'une application car elles supposent d'être à même de découper les tâches en deux parties, l'une obligatoire, fixe, parfaitement déterminée et l'autre optionnelle et de comportement plus aléatoire (par exemple, ces méthodes sont inadaptées à toutes les applications de traitement du signal où l'on ne peut se contenter de résultats approximatifs). De plus, elles compliquent la gestion de l'ordonnancement et le développement de l'application.

3.2.2. Le surdimensionnement du système

Une première méthode consiste à délibérément laisser du temps creux dans la configuration afin d'être certain que les tâches pourront entièrement s'exécuter sur un intervalle de temps. Prenons un procédé dont le contrôle est assuré cycliquement par des tâches T1, T2, T3; on s'arrange alors pour que la somme des temps moyen d'exécution de ces trois tâches soit égale à deux tiers, voire la moitié de la longueur du cycle.

Une autre façon d'agir est d'incorporer explicitement les tâches apériodiques à l'ensemble des tâches périodiques afin de pouvoir les ordonnancer sur un même pied d'égalité. Ainsi, à la place d'une tâche apériodique, le concepteur développe une tâche périodique chargée régulièrement d'examiner si le phénomène d'exception (par exemple le feu) ne s'est pas déclenché et si oui, de lancer le traitement adéquat pour y faire face (appeler les pompiers; évacuer les lieux).

Bien-sûr, ces méthodes aggravent encore le problème de sous- utilisation du processeur.

4. Quelles solutions?

Si l'on désire s'assurer un taux acceptable d'utilisation du processeur, il faut travailler sur la base de la période d'activation de "croisière" et des temps courants d'exécution des tâches. D'autre part, la suprématie accordée aux tâches périodiques doit être supprimée.

En cas de surcharge, il faut alors s'assurer qu'au moins les tâches les plus importantes sont garanties. Un algorithme présentant cette propriété est dit **stable** [26].

L'échéance d'une tâche décrit seulement l'urgence de la tâche mais ne donne pas d'indication sur son importance et malheureusement, ces deux critères, urgence et importance, ne s'accordent pas forcément. Des tâches de faible importance peuvent avoir un court délai critique et vice-versa.

Il faut donc introduire dans la politique d'ordonnancement un nouveau paramètre représentant l'importance et planifier l'exécution des tâches en tenant compte à la fois de leur laxité et de leur importance dans l'application. Lorsqu'il y a surcharge du système, le critère d'importance prévaut sur le critère d'urgence et ce sont les tâches pour lesquelles le critère d'importance est le plus fort qui sont garanties en priorité.

Plusieurs politiques ont été développées mais nous les classerons en deux courants, selon la forme prise par le critère d'importance: dans le premier [27][28], ce paramètre à la forme d'un entier; dans le second [29] [30] [31] [32], c'est une fonction du temps. Par ailleurs, le premier courant s'adresse tout à fait à des systèmes temps réel stricts tandis que le deuxième concerne les systèmes temps réel plus faiblement contraints.

Ces politiques se rapprochent dans le sens où elles s'appuient toutes sur un même principe: tant qu'il n'y a pas de surcharge, i.e. qu'une tâche ne peut

être garantie, l'ordonnancement se fait selon l'Earliest Deadline, sur la seule base des échéances. En cas de surcharge, le critère d'importance est utilisé pour éliminer des tâches.

La sémantique générale de ces ordonnancements est:

- En cas de surcharge, définir les tâches à éliminer afin de pouvoir assurer les échéances des tâches de plus haute importance. Ce point peut impliquer une relativisation de la garantie dans le sens qu'une tâche T acceptée peut ne pas le demeurer s'il survient avant la fin de son exécution une tâche T' de plus haute importance ne pouvant être garantie que si T est éliminée.
- En charge normale:
  - veiller à la détection de la surcharge avant qu'il ne soit trop tard.
  - après retour à la normale, réinsérer les tâches éliminées.

5. Le critère d'importance

5.1. Les algorithmes de [27]-[28]

L'importance d'une tâche est définie par un entier dont la valeur est fixée à l'initialisation du système et elle reste immuable au cours de la vie de celui-ci.

5.1.1 L'algorithme de [27]

Le contexte de travail est celui d'un système distribué, avec un ordonnanceur global qui se charge de répartir la charge de travail entre les sites en appliquant la politique de l'algorithme souple [33]. L'algorithme de garantie local à un site est invoqué à chaque réveil d'une nouvelle tâche T et si la tâche T ne peut être garantie localement, elle est envoyée sur un autre site susceptible, lui, de pouvoir la garantir. En regard de la sémantique donnée précédemment, le seuil de surcharge est donc défini par l'impossibilité de garantir une nouvelle tâche arrivante.

Toutes les tâches sont apériodiques.

Le principe général de l'algorithme est le suivant:

A l'arrivée d'une nouvelle tâche T

- 1. Tenter d'ordonnancer T selon la politique Earliest Deadline, sans se préoccuper de son importance IMP.
- 2. Si la tâche T ne peut être garantie, tenter de la garantir aux dépens d'autres tâches T' acceptées précédemment et de critère IMP inférieur. Deux façons d'opérer pour choisir les tâches T' à éliminer sont données:
  - le choix se fait tâche par tâche et dans l'ordre strictement croissant d'importance, i.e. en commençant par celles de plus petite importance.
  - le choix se fait tâche par tâche mais ne suit plus l'ordre strictement croissant du facteur IMP. Les premières tâches éliminées sont celles d'importance inférieure à T et de plus grande laxité.

  A chaque fois qu'une tâche T' est éliminée, l'ordonnançabilité de T doit être retestée. Comme ceci peut se révéler coûteux, une heuristique est utilisée et le test de garantie est seulement exécuté lorsque la somme des temps d'exécution des tâches T' est supérieure ou égale au temps d'exécution de la tâche T. Ceci est évidemment pessimiste dans la mesure où les temps creux existants ne sont pas pris en compte.
- 3. Si on n'a pas pu trouver assez de tâches T' à éliminer afin de garantir la tâche T, transférer la tâche T sur un autre site.

Le contexte de travail étant celui d'un système distribué, les tâches T' sont envoyées à d'autres sites dans l'espoir qu'elles pourront y être garanties. Un processus, identique à celui décrit ci-dessus, est effectué pour chacune d'elle sur le site destinataire.

5.1.2 L'algorithme de [28]

Le principe est très similaire à celui de l'algorithme développé par Biyabani et en fait, ne diffère de celui-ci que sur un point: le contexte est centralisé et chaque tâche dispose d'un "suppléant", c'est-à-dire d'une tâche qui réalise un service équivalent, mais plus rapidement et d'une manière évidemment dégradée (en quelque sorte, c'est un secondaire). Lorsqu'en période de surcharge, une tâche est supprimée, son suppléant

est activé et son exécution est prise en compte, i.e. si le temps d'exécution de la tâche est C et celui de son suppléant c, on considère ne libérer que C - c unités de temps processeur.

5.2. L'ordonnancement conduit par fonction du temps [29] [30] [31] [32]

5.2.1 Le critère d'importance comme fonction du temps

L'approche de ces auteurs est différente dans leur façon de représenter le poids d'une tâche au sein de l'application à laquelle elle appartient. Le moment auquel une tâche produit ses résultats a une grande importance vis-à-vis de la justesse de l'application, au sens où un résultat survenant trop tôt ou trop tard peut produire un effet plus néfaste qu'un résultat ne survenant pas du tout. L'achèvement d'une tâche possède donc pour une application temps réel une valeur variant dans le temps, reflétant le degré d'adéquation du moment auquel la tâche se termine en produisant ses données ou en achevant son action sur le procédé contrôlé. L'approche courante pour représenter cette valeur est de fixer une échéance D à la tâche signifiant par la même que le résultat sera incorrect s'il survient au delà d'une certaine limite de temps. Selon l'impact qu'engendre sur le procédé un dépassement de cette date D, on autorise plus ou moins de retard, définissant ainsi des tâches à échéances strictes et des tâches à échéances relatives, avec généralement aucune précision sur la relativité acceptée. Ce schéma peut se révéler mal adapté pour représenter les contraintes temporelles de certaines tâches, notamment celles précisant qu'une tâche ne doit produire ses résultats que dans un intervalle entourant D, i.e. pas trop tard, mais **pas trop tôt non plus** (par exemple, sur une chaîne de montage: si un bras robotisé doit saisir des objets défilant devant lui, il ne doit pas tenter de le faire trop tôt car l'objet n'est pas encore arrivé, ni trop tard car l'objet est passé). Les auteurs, souhaitant synthétiser cette méthode courante et remédier à ses points faibles ont choisi de marier la spécification de l'échéance d'une tâche et de son importance au moyen d'une fonction du temps V, indiquant pour tout t, la valeur pour l'application de la terminaison de la tâche[12]. L'échéance de la tâche T est représentée par une discontinuité dans la courbe de la fonction V(t); si celle-ci est stricte, la fonction au temps D passe à une valeur nulle, sinon elle décroît plus ou moins lentement. Une tâche s'achevant avec une valeur positive offre un résultat satisfaisant pour l'application; dans le cas contraire, il peut survenir des dommages pour celle-ci.

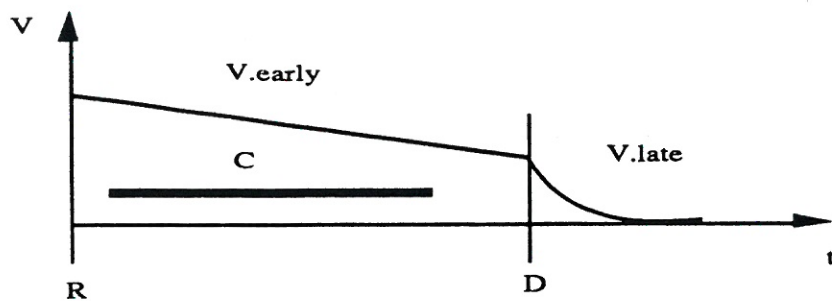
La définition pour chaque tâche de sa fonction V est réalisée à l'aide d'informations concernant à la fois l'environnement extérieur et le système. Il est à noter qu'elle ne doit pas toujours être facile à effectuer. Des exemples sont fournis dans [30]: pour une chaîne de fabrication, il propose de se baser sur la valeur commerciale des produits fabriqués; pour des applications militaires, sur le nombre de vies humaines sauvées ou sur l'importance stratégique des objectifs atteints. Les fonctions V sont divisées en deux parties, la partie précédant l'échéance (V.early) et la partie lui succédant (V.late). Chaque partie a pour forme générale $V(t) = K_1 + K_2t + K_3t^2 + K_4e^{-K_5t}$, ce qui permet de définir des fonctions constantes, linéaires, quadratiques, exponentielles ou des fonctions linéairement composées de ces premières formes.

La Figure 20 donne des exemples de ces fonctions du temps:

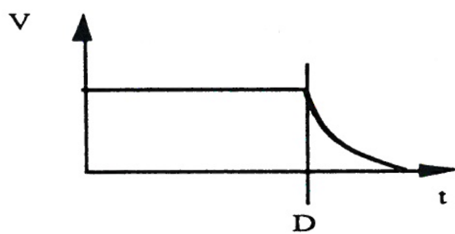
La fonction CAS_1 est formée d'une partie V.early constante et d'une partie V.late décroissant exponentiellement: cette forme peut s'appliquer à un calcul opéré cycliquement dont le résultat peut être accepté tant que le cycle suivant ne doit pas commencer.

La fonction CAS_2, fournit la représentation d'une tâche à échéance stricte. Passée la date D, le résultat n'a plus à être produit.

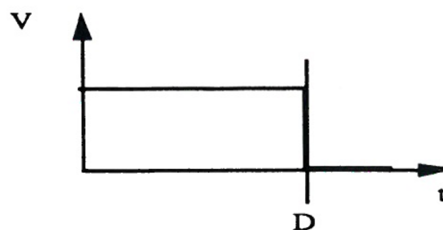
12 Une tâche est donc caractérisée par le triplet (r,C,V) où r et C sont respectivement toujours la date d'occurrence de la tâche et son pire temps d'exécution.



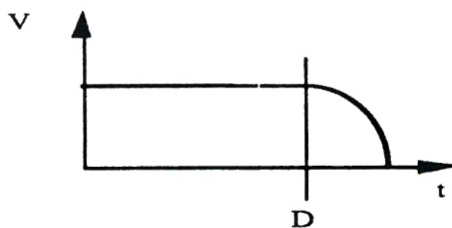
Forme générale de la fonction V pour une tâche T de temps d'exécution C, d'échéance D et de date d' occurrence R



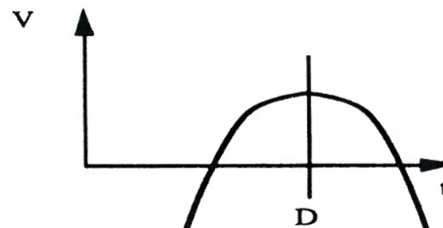
CAS 1



CAS 2



CAS 3



CAS 4

Figure 20. Exemples de fonctions V

La fonction CAS_3, avec une décroissance parabolique pour V.late,présente une tâche à échéance relative.

La fonction CAS_4, de forme parabolique pour V.early et V.late, décrit une tâche pour laquelle le résultat ne doit être produit que dans un certain intervalle de temps entourant D, i.e. pas trop tôt, ni trop tard. C'est par exemple une tâche de mise en orbite d'un satellite: celle-ci ne peut se produire que lorsqu'une certaine altitude est atteinte.

5.2.2 Principe des algorithmes

L'algorithme d'ordonnancement développé par Jensen et travaillant avec ces fonctions, de façon similaire aux algorithmes de [27], pratique une politique Earliest Deadline lorsque le système n'est pas surchargé. Cependant lorsqu'une surcharge survient, il change de tactique de travail afin de pouvoir contrôler le choix des tâches qui ne respecteront pas leurs échéances. Jusqu'à ce que la situation redevienne normale, l'ordonnanceur élimine les tâches pour lesquelles le temps d'exécution résiduel est plus grand que le temps existant jusqu'à la date d'échéance, soit les tâches pour lesquelles le rapport (valeur de V en fin d'exécution, temps d'exécution restant) est le plus petit. Le but de l'algorithme est de maximiser β, somme des valeurs V.

Régulièrement, une estimation de la charge du système est effectuée et comparée à un seuil qui détermine s'il y a surcharge ou pas. Les auteurs ne donnent pas d'indication sur la façon dont est effectuée cette estimation, et se contentent de souligner qu'elle est un peu coûteuse. Ils remarquent par ailleurs que la valeur du seuil affecte également les performances.

Les algorithmes de Baruah, Koren s'appuient sur le même principe mais diffèrent de celui-ci par leur plus grande complexité au niveau des choix des tâches à éliminer. Cette aggravation de la complexité est due au souci de doter ces algorithmes travaillant avec des fonctions du temps, d'un critère d'évaluation des performances. Soit un algorithme A_C, dit algorithme clairvoyant, qui par connaissance de toutes les données sur les tâches, effectue les suppressions optimales. On dit qu'un algorithme A possède un facteur compétitif (competitive factor) r, $0 \leq r \leq 1$, si et seulement si il est certifié qu'il atteindra une valeur B au moins égale à r fois la valeur B_C obtenue grâce à l'algorithme clairvoyant A_C.

5.3. Discussion des qualités et des défauts de ces algorithmes

Aucune des deux approches que nous venons de décrire n'est vraiment satisfaisante, d'une part à cause des hypothèses sur lesquelles elles s'appuient, d'autre part dans leur définition même.

L'approche de [29] & Co est attirante parce qu'elle permet d'exprimer le poids d'une tâche au sein de son application temps réel avec une plus grande modularité que les méthodes habituelles; notamment, elle offre la possibilité d'exprimer des contraintes de temps plus variées que ne le permet une simple date d'échéance. Cependant, le critère de choix des tâches éliminées en cas de surcharge (minimum du rapport V_i/C_i) et la volonté finale de l'ordonnancement, à savoir maximiser le facteur β, ne donnent pas entière satisfaction. Le critère choisi pour mesurer l'efficacité de l'ordonnancement (maximiser β) ressemble à une évaluation des performances en terme de maintien d'un bon rendement de l'application en surcharge (pas d'écroulement) et adresse plus difficilement les situations critiques où il s'agit de faire face à une alarme. Les exemples donnés dans [30] pour fixer les valeurs V_i (prix de l'objet fabriqué par la tâche, importance stratégique de l'opération militaire gouvernée par le processus) abondent dans ce sens et en cas de surcharge, impliquant par exemple trois ou quatre activités, la tâche de plus haute importance V peut ne pas être ordonnancée si les autres tâches de valeur individuelle moindre fournissent une valeur combinée supérieure à V. Si la tâche d'importance V devait traiter une urgence, apporter remède à un incident survenu dans le procédé, le choix est mauvais car l'application est laissée en péril. Les cas de surcharge envisagés ici découlent plus de conflits dus à des variations des caractéristiques des tâches (temps d'exécution plus importants qu'escomptés, modification de période) qu'à la venue soudaine de tâches supplémentaires.

[27] & Co, en cherchant à toujours garantir la dernière tâche survenue de plus haute importance, adresse tout à fait le problème d'un traitement d'urgence. Cependant, l'hypothèse de l'apériodicité de toutes les tâches, restreint l'applicabilité de la politique car elle ne se préoccupe pas de certains problèmes pouvant avoir trait à l'évincement de requêtes de tâches périodiques. Dans les algorithmes de Biyabani, une tâche ne pouvant pas (ou plus) être garantie sur un site est simplement envoyée sur un autre site ou bien, s'il n'a pas été trouvé de site pouvant l'accueillir, est abandonnée. Bien sûr, une tâche périodique peut être transformée, en considérant chacune de ses requêtes aux instants kP, en une série de tâches apériodiques. Mais, les tâches apériodiques ainsi obtenues sont particulières dans le sens où il existe entre elles une relation d'instanciation d'une même tâche périodique et pour chacune d'elles, les décisions d'élimination peuvent ne pas être aussi simple qu'en ce qui concerne une véritable tâche apériodique. En effet, une tâche périodique peut supporter de ne pas s'exécuter x fois de suite, mais pas x+1 fois. Illustrons ceci à l'aide du robot ASV, la tâche BMP et la tâche LS. LS peut manquer une fois son exécution mais pas deux fois de suite, car sinon le circuit hydraulique de la patte n'est plus maintenu et le robot devient instable. Ainsi, un coup sur deux, l'importance de BMP devient inférieure à celle de LS. L'importance de la tâche doit donc varier au cours de la vie de l'application afin d'empêcher qu'elle soit toujours choisie pour être supprimée. D'autre part, il est facile d'imaginer que l'importance des tâches peut également varier en fonction du contexte d'évolution de l'application. Cette variation de l'importance des requêtes des tâches périodiques en fonction du nombre de fois où elles ont déjà été éliminées, de l'importance de toutes sortes de tâches en fonction de l'environnement, n'est envisagée dans aucune des méthodes que nous avons présentées.

Par ailleurs, ces importances variables amènent bien le besoin d'un mécanisme spécial de réinsertion de tâche, pour permettre la réintroduction dans la séquence d'ordonnancement des tâches éliminées dont l'exécution devient obligatoire pour la survie du système.

Le dépassement d'échéance par une tâche de l'application temps réel est souvent pris comme indice de surcharge et facteur déclenchant de la politique d'ordonnancement selon le critère d'importance. C'est la cas de [27] et du système Haliotis [34]. Ceci a certainement besoin d'être nuancé: le changement de politique d'ordonnancement, les mécanismes de choix des tâches, de suppression de celles-ci ne sont pas gratuits et les appliquer systématiquement dès qu'une tâche viole ou risque de violer son échéance n'est peut-être pas très judicieux. Si la tâche en péril est de faible importance, si elle fait partie du lot des tâches identifiées comme pouvant éventuellement sauter une(ou plusieurs) exécution(s) ou dépasser son échéance avec une certaine amplitude, il peut être plus simple et pas beaucoup plus gênant de laisser aller.

L'événement amenant la prise en compte de l'importance des tâches et le basculement vers une politique de suppression de tâches en exécution ou précédemment acceptées doit donc être plus fin qu'un simple dépassement d'échéance. On peut alors penser à définir un seuil de charge, comme le fait [29]. Le problème est qu'un seuil de charge employé seul ne résout pas non plus toutes les difficultés: qu'il survienne une tâche apériodique de haute importance ne pouvant respecter ses contraintes de temps qu'aux dépens d'autres tâches de moindre importance, mais que la charge n'augmente pas suffisamment pour dépasser le seuil, alors la tâche apériodique est condamnée et l'application avec. Il ressort de là que la définition du seuil doit être le fruit d'une réflexion vigilante, qui en prenant en compte les données du système hôte et les exigences de l'application, assure le déclenchement de mécanismes de suppression en vue de la garantie des tâches de traitement d'urgence et du maintien d'un bon rendement. Se pose cependant le problème du coût de l'évaluation de ce seuil en cours de la vie du système temps réel [29], sans donner plus de détails sur sa façon d'opérer, souligne les pénalités dont le coût est cause.

Une autre solution peut être de combiner les deux approches, dépassement d'échéance et surveillance de la charge en modulant l'impact de la première par l'importance de la tâche dont l'échéance risque d'être violée.

L'inconvénient de ce style de politique qui conditionne son action sur un seuil unique et fixe est le risque de "jouer au yo- yo" avec le seuil, i.e. de sans cesse passer d'un côté et de l'autre. Une stratégie modelée sur celle utilisée dans [34] est attirante: chaque tâche possède un rang de suppression ρ et la valeur de la charge conditionne la valeur du rang de suppression critique ρ_o. Toutes les tâches dont le rang de suppression ρ est inférieure à ρ_o sont éliminées et réinsérées lorsque leur rang sera redevenu supérieur à ρ_o. Ceci permet une dégradation plus étalée des exécutions et une réinsertion plus graduelle, donc mieux maîtrisée.

Un autre point que les auteurs n'abordent pas est celui des conditions sous lesquelles une tâche peut être éliminée. Ceci doit se faire sans mal pour les autres, pour les ressources et les données partagées. Il faut veiller à laisser l'environnement d'exécution, le procédé dans un état cohérent et sûr. Il est alors fortement possible que certaines tâches ne soient pas stoppables.

6. Conclusion

Les algorithmes d'ordonnancement temps réel jusqu'à présent développés travaillent sous des hypothèses qui ne sont pas vérifiées dans le monde concret des applications temps réel ou qui y sont mal adaptées. Les temps d'exécution et les fréquences d'activation en réalité variables, les retards engendrés par les conflits d'accès aux ressources, amènent par l'emploi des algorithmes traditionnels des dépassements d'échéances imprévisibles et de mauvaises décisions d'ordonnancement. Les méthodes utilisées pour remédier à ces problèmes - rendre la machine prévisible, remaniement des applications, mécanismes se basant sur plusieurs versions de tâches- ne suffisent pas ou ne sont pas applicables à toutes les tâches ou ajoutent un coût non négligeable à l'ordonnancement.En fait, un critère d'ordonnancement basé sur l'urgence des tâches ne suffit pas; il est nécessaire de lui ajouter un critère basé sur l'importance de la tâche au sens du poids de celle-ci dans l'application.

Ainsi, en cas de surcharge, le critère d'urgence qui ne peut plus être correctement appliqué, est abandonné et les tâches de plus grande importance sont garanties en priorité sur des tâches de moindre importance. Cette nouvelle perception du problème de l'ordonnancement, de plus, adresse de façon plus satisfaisante, la prise en compte des tâches apériodiques, tâches généralement déclenchées lors de l'occurrence d'une situation d'exception au sein du procédé contrôle: ces tâches, plus importantes que les tâches périodiques réalisant le suivi du procédé vu qu'elles ont pour objectif d'entreprendre des actions de sauvegarde du procédé,seront garanties au détriment des tâches périodiques, le temps que l'état du procédé soit redevenu correct.

Nous avons étudié plusieurs politiques utilisant ainsi un critère d'importance en plus du critère d'urgence pour ordonnancer les tâches: aucune d'elles n'est vraiment satisfaisante: soit elles n'adressent pas bien le problème de maintien du procédé dans un état correct en surcharge, soit elles sont trop rigides et ne prennent pas en compte tous les cas de Figures.

Ces différents cas de Figures sont:

- la gestion des phases de l'application,
- les tâches essentielles dont certaines requêtes peuvent dépasser leur échéance,
- les tâches périodiques pour lesquelles un certain nombre de requêtes peuvent être non exécutées ou fautives,
- l'importance des tâches variant selon le passé de l'application ou le contexte d'exécution,
- le problème du maintien de l'intégrité du système lorsque certaines requêtes sont arrêtées en cours d'exécution ou non exécutées.

Il semble donc que l'une des voies de recherche en ordonnancement temps réel des années à venir soit l'intégration de ces cas de Figures afin de se rapprocher davantage des problèmes auxquels se heurtent les concepteurs d'applications temps réel.

BIBLIOGRAPHIE

1. **CNRS: Groupe de réflexion temps réel du CNRS**, Le temps réel, TSI, Vol. 7-5, 1988, pp. 493-500.
2. SCHWAN,K., BO,W. and GOPINATH,P.,**A High-Performance Object- Based Operating System for Real-Time, Robotics Applications**, Proceedings of Real-Time Systems Symposium, 1986, pp. 147-156.
3. PAYTON,D.W. and BIHARI,T.E., **Intelligent Real-Time Control of Robotics Vehicles**, COMMUNICATIONS OF THE ACM, août 1991, Vol. 34, No. 8, p. 4963.
4. STANKOVIC,J.A. and RAMAMRITHAM, K., **What is Predictability for Real-Time Systems?** JOURNAL OF REAL-TIME SYSTEMS, Vol. 2, 1990, pp. 247-254.
5. SPRUNT,B., SHA,L. and LEHOCZHY,J.P., **Aperiodic Task Scheduling for Hard Real-Time Systems**, JOURNAL OF REAL-TIME SYSTEMS, 1989, pp.27-60.
6. PAUL,C.J., ACHARYA,A., BLACK,B.and STROSNIDER,J.K.,**Reducing Problem-solving Variance To Improve Predictability**, COMMUNICATIONS OF THE ACM, Vol. 34, No.8, août 1991,pp. 81-93.
7. CHETTO,H., **L'ordonnancement dans les systèmes de contrôle temps réel à contraintes strictes**, Thèse de Doctorat d'Etat, Université de Nantes, Ecole Nationale Supérieure de Mécanique, décembre 1990, 192 p.
8. LEUNG,J.Y.T. and MERILL,M.L., **A Note on Preemptive Scheduling of Periodic Real-Time Task**, INFORMATION PROCESSING LETTERS, 20(3), 1980, pp. 115-118.
9. LIU,C.L.and LAYLAND,J.W., **Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment**, JOURNAL OF ACM, Vol. 20, No. 1, janvier 1973, pp. 46-61.
10. SORENSON,P.G., **A Methodology for Real-Time System Development**,Ph.D Thesis, University of Toronto, Canada, 1974.
11. DHALL,S.K.,**Scheduling Periodic-Time Critical Jobs on Single Processor and Multiprocessor Computing Systems**, Ph.D Thesis, University of Illinois, avril 1977.
12. SILLY,M. and CHETTO,H.,**How to Achieve Optimal Responsiveness in Semi-Hard Real-Time Environments**, Rapport Interne LAN-ENSM No. 91-15, mai 1991, 20 p.
13. ROUX,O.H.,**Ordonnancement dans les systèmes temps réel semi- stricts**,Université de Nantes, Ecole Centrale de Nantes, Laboratoire d'Automatique, Rapport de DEA, 24 septembre 1991, 76 p.
14. CHETTO,H. et DELACROIX, J., **Minimisation des temps de réponse des tâches sporadiques en présence des tâches périodiques**, RTC'93, Paris,12-15 janvier 1993.
15. LEINBAUGH,D.W., **Guaranteed Response Times in a Hard-Real-Time Environment**, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. SE- 6, No. 1, janvier 1980, pp. 85-91.
16. DORSEUIL, A. et PILLOT,A., **Le temps réel en milieu industriel: coneept environnement multitâches, DUNOD, 1991, 296 p.**
17. KAISER,C., **Exclusion mutuelle et ordonnancement par priorité**, TSI, Vol. 1-1, 1982 , pp.59-69.
18. MINET,P., **Evaluation de performances des protocoles temps réel GAM-T-103**,TSI, Vol. 9, 1990, pp. 5-17.
19. ROLLIN,P., **Mesure de systèmes répartis et réseaux**, Thèse de Doctorat d'Etat, Université de Rennes, octobre 1987.
20. LE LANN,G., **Le projet SCORE: les systèmes temps réel**, TSI, Vol. 6, No. 2, 1987, pp. 175-178.
21. ZHAO,W. and RAMAMRITHAM,K., **Virtual Time CSMA Protocols for Hard Real-Time Communication**, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. SE-13,No. 8, août 1987, pp. 938-952.
22. MARCE-MARONDO,J., **Un simulateur d'ordonnancement de tâches temps réel, Approche orientée objets, langage Ada**, Mémoire d'Ingénieur CNAM en informatique, mars 1993, 127 p.
23. CAMPBEL, R.H., HORTON,K.H. and BELFORD,G.G.,**Simulations of a Fault Tolerant Deadline Mechanism**, Digest of papers FTCS-9, 1979, pp.95-101.
24. ELYOUNSI,N., **Ordonnancement et reconfiguration dynamique dans un système temps réel réparti à contraintes strictes**, Université de Nantes, Ecole Nationale Supérieure de Mécanique, Thèse de Docteur

de 3ème cycle, 1991, 200 pages, TSI, Vol. 7-5, 1988, pp. 493-500.

25. CHUNG,J.Y.and LIU, J.S., **Algorithms for Scheduling Periodic Jobs To Minimize Average Error**, Proc. Real-Time Sys. Symp., Huntsville, Alabama, USA, décembre 1988, pp. 142-151.
26. SHA,L., LEHOCZHY,J.P. and RAJKUMAR,R., **Solutions for Some Practical Problems in Prioritized Preemptive Scheduling**,Proceedings IEEE International Conference on Real-Time Systems, mai 1986, pp. 181-191.
27. BIYABANI,S.R., STANKOVIC,J.A. and RAMAMRITHAM,K., **The Integration of Deadline and Criticalness in Hard Real-Time Scheduling**,Proceedings IEEE Conference on Distributed Computing Systems,1988, pp.152-159.
28. MILLER,F.W.,**The Performance of a Mixed Priority Real-Time Scheduling Algorithm**, ACM OPERATING SYSTEMS REVIEW, Vol. 26, No. 4, octobre 1992, pp. 5-13.
29. JENSEN,E.D., LOCKE,C.D. and TOKUDA,H., **A Time-Driven Scheduling Model for Real-Time Operating Systems**, Proceedings of 1985 IEEE Real-Time Systems Symposium, 1985, pp. 112-122.
30. CLARK,R.K., **Scheduling Dependent Real-Time Activities**, Ph.D Thesis, Carnegie-Mellon University, mai 1990, 240 p.
31. BARUAH,S., KOREN, G., MISHRA, B., RAGHUNATHAM,A., ROSIER,L.and SHASHA, D.,**On - line Scheduling in the Presence of Overload**, IEEE Foundations of Computer Science Conference, San Juan, Puerto Rico, octobre 1991,pp.101-110.
32. KOREN, G. and SHASHA,D.,**D-OVER: An Optimal On-line Scheduling Algorithm for Overloaded Real-Time Systems**,Rapport technique de l'INRIA-Rocquencourt, No. 138, février 1992, 45 p.
33. STANKOVIC,J.A., RAMAMRITHAM, K. and CHENG,S.,**Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems**, IEEE TRANSACTIONS ON COMPUTERS, Vol. C-34, No. 12, décembre 1985, pp. 1130-1143.
34. DERVILLE, KAISER, PEIROTES et TELLIER,**Le système HALIOTIS**, R.I.R.O, 1ère année, No. 6, 1967, pp. 3-25.