Chapitre II

Ordonnancement monoprocesseur de tâches temps-réel: état de l'art et contributions

1. Introduction

Le premier objectif de ce chapitre est, tout d'abord de présenter des algorithmes qui garantissent le respect des contraintes temporelles au moyen d'une fonction de garantie ou *test d'ordonnançabilité*. Le test d'ordonnançabilité doit indiquer, par un calcul si possible simple et rapide, si l'ensemble des tâches peut être ordonnancé par l'algorithme et ce sans avoir besoin d'exécuter l'algorithme.

Il est plus particulièrement essentiel:

- d'une part, de rappeler les principaux algorithmes préemptifs optimaux au sens de la garantie des contraintes temporelles (un algorithme est dit *optimal*, pour des hypothèses données, si aucun autre algorithme (pour les mêmes hypothèses) n'est capable d'ordonnancer une configuration de tâches qui ne soit pas ordonnançable par cet algorithme); ces algorithmes sont les algorithmes les plus utilisés et sont la base de toutes les études actuelles sur l'ordonnancement.
- d'autre part, de présenter les principaux travaux existants, à notre connaissance, sur les algorithmes non-préemptifs. Dans le contexte de notre travail sur l'ordonnancement de messages temps-réel à travers une ressource de transmission (la transmission est non-préemptive par définition), ces algorithmes sont à considérer très particulièrement; notons que les algorithmes non-préemptifs présentés sont des versions non-préemptives des algorithmes préemptifs (d'où l'intérêt d'avoir présenté au préalable les algorithmes préemptifs).

Un second objectif de ce chapitre est précisément de présenter les travaux que nous avons faits sur les algorithmes non-préemptifs.

En conséquence, ce chapitre comprend trois paragraphes:

- dans un premier paragraphe, nous résumons l'état-de-l'art sur les principaux algorithmes préemptifs, c'est-à-dire, l'ordonnancement de tâches périodiques, l'ordonnancement conjoint de tâches périodiques et apériodiques, ainsi que la

problématique du partage de ressources (le problème des surcharges de fonctionnement n'est pas considéré);

- dans un deuxième paragraphe, nous présentons les principaux travaux sur les algorithmes non-préemptifs;
- dans un troisième paragraphe, nous présentons notre contribution au domaine des algorithmes non-préemptifs;

2. Algorithmes préemptifs

2.1. Tâches périodiques

Les algorithmes présentés sont basés sur la considération du cas pire, c'est-à-dire, ils évaluent l'ordonnançabilité d'un ensemble de tâches périodiques à un *instant critique*, un instant critique étant un instant où le temps de réponse de l'ordonnanceur à la demande d'exécution de toutes les tâches est le plus long.

L'instant critique d'une tâche [LL, 73] est l'instant où la demande d'exécution d'une tâche arrive simultanément avec la demande d'exécution de toutes les tâches de priorité supérieure. Le cas pire est donc considéré si on suppose que les déphasages entre les tâches sont nuls: il suffit alors de vérifier si la première demande de chaque tâche est exécutée avant son échéance.

Nous résumons successivement les cas des algorithmes (c.f. chapitre I) à échéance sur requête ($d_i = P_i$) et à échéance avant requête ($d_i < P_i$).

2.1.1. Échéance sur requête

Les travaux fondamentaux sur les algorithmes à échéance sur requête ont été faits par Liu et Layland [LL, 73] et concernent les algorithmes "Rate Monotonic" (RM) et "Earliest Deadline" (ED) dont nous présentons maintenant les éléments principaux.

1. Algorithme RM

C'est un algorithme qui est basé sur des priorités statiques (une tâche a une priorité fixe qui est inversement proportionnelle à sa période) et qui est optimal.

Test d'ordonnançabilité

- **Condition suffisante [LL, 73]:**

  L'ordonnancement RM d'un ensemble de n tâches périodiques $t = \{t_1, \ldots t_i, \ldots t_n\}$ est faisable si le facteur d'utilisation $U = \sum_{i=1}^{n} C_i / P_i$ vérifie la relation suivante:

  $$U \leq n \cdot \left(\sqrt[n]{2} - 1\right) \qquad (2.1)$$

  Compte tenu que cette condition n'est que suffisante, il ne faut pas écarter des ensembles de tâches qui ne satisfont pas ce test. Des travaux [LSD, 89] ont présenté une condition nécessaire et suffisante d'ordonnançabilité.

- **Condition nécessaire et suffisante [JP, 86]:**

  Considérons un ensemble de n tâches périodiques $t = \{t_1, \ldots t_i, \ldots t_n\}$ ordonnancé par RM; sachant que le temps de réponse d'une tâche t_i est borné supérieurement par

$RT_i = \left(\sum_{j=1}^{i-1} \lceil RT_i / P_j \rceil \cdot C_j \right) + C_i$, l'ordonnancement de l'ensemble t est faisable si et seulement si:

$$\forall_i,\ 1 \le i \le n,\ \ RT_i \le P_i \tag{2.2}$$

Exemple

Considérons un ensemble de 3 tâches $t = \{t_1, t_2, t_3\}$, avec $(C_i, P_i) = \{(1,5),(3,6),(5,100)\}$, à ordonnancer par l'algorithme RM. Comme la condition suffisante d'ordonnançabilité est satisfaite $\left(U = 0.75 < 3 \cdot \left(\sqrt[3]{2} - 1\right) = 0.78\right)$, l'ensemble de tâches est ordonnançable. La séquence temporelle d'exécution des tâches après l'instant critique est représentée sur la figure 2.1.

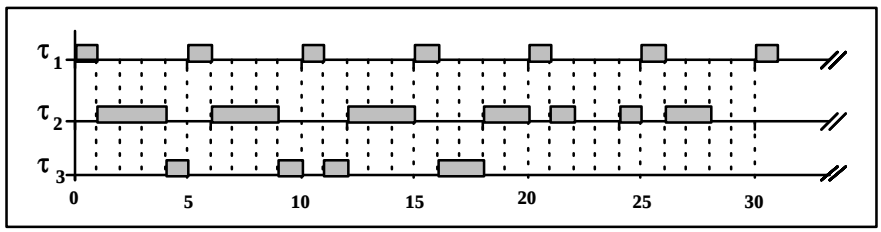

Figure 2.1: Exemple d'ordonnancement RM

La tâche t_1 est toujours exécutée au début de la période, car cette tâche est toujours la tâche la plus prioritaire (algorithme à priorité statique). La tâche moins prioritaire (t_3) est toujours préemptée pour permettre l'exécution d'une tâche plus prioritaire (t_1, t_2).

2. Algorithme ED

C'est un algorithme qui est basé sur des priorités dynamiques (à toute instant, la priorité est inversement proportionnelle à la date de l'échéance) et qui est optimal. L'algorithme ED permet d'accepter un plus grand nombre de configurations de tâches que l'algorithme RM.

Test d'ordonnançabilité (condition nécessaire et suffisante)

L'ordonnancement ED d'un ensemble de n tâches périodiques $t = \{t_1, \ldots t_i, \ldots t_n\}$ est faisable si et seulement si le facteur d'utilisation $U = \sum_{i=1}^{n} C_i / P_i$ vérifie la relation suivante:

$$U \le 1 \tag{2.3}$$

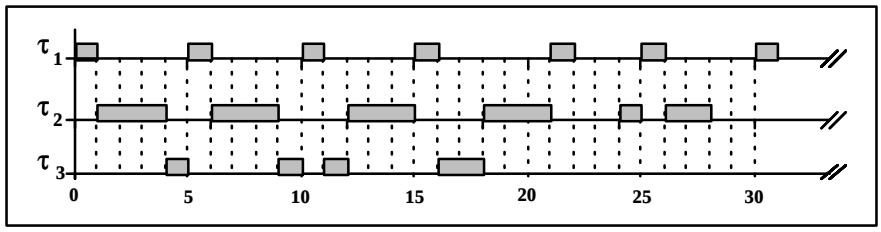

Figure 2.2: Exemple d'ordonnancement ED

Exemple

Considérons le même ensemble de 3 tâches périodiques: $(C_i, P_i) = \{(1,5),(3,6),(5,100)\}$, dans le cas d'un ordonnancement ED. Comme le test d'ordonnançabilité est vérifié, l'ensemble des tâches est ordonnançable ($U = 0.75 \leq 1$). La séquence temporelle d'exécution des tâches après l'instant critique est représentée dans la figure 2.2.

Dans ce cas, comme l'algorithme ED est un algorithme à priorité dynamique, la tâche t_1 n'est pas toujours la plus prioritaire, et donc ses travaux ne sont plus systématiquement exécutés au début de la période. A chaque instant, le travail en attente à exécuter sera celui dont la date d'échéance est la plus proche.

2.1.2. Échéance avant requête

Un des inconvénients de l'algorithme RM est celui de considérer les tâches à échéance sur requête, c'est-à-dire, $d_i = P_i$.

Il s'avère intéressant de considérer aussi le cas des tâches avec des échéances inférieures à la période: $d_i < P_i$. Supposons par exemple un système avec une tâche déclenchée toutes les 10s qui doit se terminer dans les 10ms qui suivent. Un ordonnancement qui utilise l'algorithme RM ou ED est complètement inefficace, parce que l'accomplissement de l'exécution des travaux de la tâche n'est garanti qu'avant la fin de sa période, c'est-à-dire 10s.

1. Algorithme "Deadline Monotonic" (DM)

C'est un algorithme qui est basé sur des priorités statiques (une tâche a une priorité fixe qui est inversement proportionnelle à son échéance). Cet algorithme est équivalent à l'algorithme RM dans le cas de tâches à échéance sur requête.

Cet algorithme n'a vraiment été utilisé que depuis qu'une condition suffisante d'ordonnançabilité a été présentée dans [AB, 90].

Test d'ordonnançabilité (condition suffisante)

L'ordonnancement DM d'un ensemble de n tâches périodiques $t = \{t_1, \ldots t_i, \ldots t_n\}$ est faisable si:

$$\forall i: 1 \leq i \leq n: \left\{ C_i + \sum_{j=1}^{i-1} \lceil d_i / P_j \rceil C_j \leq d_i \right\} \qquad (2.4)$$

Dans ce test d'ordonnançabilité, $\left\{ \sum_{j=1}^{i-1} \lceil d_i / P_j \rceil C_j \right\}$ est une mesure de l'interférence des processus de plus haute priorité dans l'exécution de t_i. Ce test énonce que pour une tâche t_i, la somme de la durée de son travail, plus l'interférence dans son exécution qui lui est imposée par les travaux des tâches de priorité supérieures, ne doit jamais dépasser son échéance.

La condition énoncée est suffisante mais pas nécessaire, parce que la mesure de l'interférence effectuée est pessimiste dans le sens qu'elle ne prend pas en compte les instants de déclenchement des différentes tâches.

2. Algorithme "Earliest Deadline" (ED)

Il a été montré [LM, 80] qu'une configuration arbitraire de tâches périodiques, avec échéance avant requête, est ordonnançable par l'algorithme ED, si:

$$\sum_{i=1}^{n} C_i / d_i \leq 1 \qquad (2.5)$$

Cette condition est une condition suffisante (mais non nécéssaire) d'ordonnançabilité.

2.1.3. Ordonnancement des tâches apériodiques

On dispose d'un ensemble de tâches périodiques qui ont leurs échéances garanties (c.f.: II.2.1.1 et II.2.1.2). Le problème est d'ordonnancer des tâches apériodiques sans remettre en cause l'ordonnançabilité des tâches périodiques:

- pour les tâches apériodiques à contraintes relatives il s'agit de minimiser le temps de réponse;
- pour les tâches apériodiques à contraintes strictes il s'agit de maximiser le nombre de tâches qui peuvent être acceptées;

Nous résumons les principales techniques [Mar, 94] [Del, 94]:

1. Tâches apériodiques à contraintes relatives

Les techniques les plus importantes sont la technique du *serveur de tâches* et la technique de *l'ordonnancement conjoint*. Les tâches périodiques sont ordonnancées avec l'algorithme RM, quand on utilise la technique du serveur des tâches, et avec l'algorithme ED quand on utilise la technique de l'ordonnancement conjoint.

Le serveur des tâches

Le serveur est une tâche périodique, généralement de la plus haute priorité, qui est créé pour ordonnancer les tâches apériodiques.

On a plusieurs types de serveurs qui diffèrent par la manière dont ils répondent aux arrivées des travaux: traitement par scrutation, serveur ajournable, serveur sporadique (on peut se reporter aux références [LSS, 87] et [SSL, 89])

L'ordonnancement conjoint

L'idée est de transformer les tâches apériodiques à contraintes relatives en tâches apériodiques à contraintes strictes, en ajoutant à leurs paramètres temporels une échéance fictive. Cette échéance fictive est calculée chaque fois qu'un travail d'une tâche apériodique survient et correspond à l'échéance minimale que l'on doit associer à la tâche pour qu'elle s'exécute avec un temps de réponse minimal.

L'échéance fictive est la première date pour laquelle le temps total d'inactivité du processeur, qui doit ordonnancer, dans le respect de leurs échéances, les travaux des tâches périodiques et tâches apériodiques précédemment déclenchés et non encore achevés, est égal au temps d'exécution de la tâche [CC, 89].

2. Tâches apériodiques à contraintes strictes

On a deux techniques principales: la technique de la *pseudo-période* et la technique du *test de garantie*.

Pseudo-période

On associe aux tâches apériodiques une pseudo-période qui représente l'intervalle minimal entre deux occurrences successives d'un travail d'une tâche apériodique. On ne travaille donc qu'avec un seul modèle de tâches (tâches périodiques) que l'on sait bien traiter avec les algorithmes RM ou ED. Cette technique induit cependant une sous utilisation du processeur (du fait du choix de l'intervalle minimal comme pseudo-période).

Test de garantie

Cette technique, contrairement à la précédente, ne fait aucune supposition sur la fréquence d'arrivée des tâches qui peuvent survenir à tout instant. Plusieurs modalités peuvent être envisagées. Citons, en particulier, la modalité où on teste si la nouvelle requête apériodique peut être placée dans le temps creux d'une configuration périodique ordonnancée selon l'algorithme ED et dans le respect des échéances des tâches périodiques et des tâches apériodiques précédemment acceptées [CC, 89].

2.1.4. Partage de ressources

Le partage de ressources en mutuelle exclusion peut engendrer des inversions de priorités (une tâche utilisant une ressource bloque momentanément une tâche plus prioritaire qui voudrait utiliser la ressource) ou des interblocages. Ceci se produit lorsque l'ordonnancement des tâches pour l'accès au processeur est explicitement distinct de l'ordonnancement pour l'accès aux ressources. Afin de limiter les deux phénomènes précités, des techniques qui considèrent de façon globale l'ordonnancement pour l'accès au processeur et aux ressources ont été introduites: les techniques par héritage [SRL, 90]. On distingue principalement la technique à priorité hérité et la technique à priorité plafond.

1. La priorité héritée

Principe

Lorsqu'une tâche de faible priorité t_{k+1} bloque, pour l'utilisation d'une ressource, des tâches de plus forte priorité $t_1,\dots t_k$, alors, le travail de la tâche t_{k+1} s'exécute avec une priorité égale à $max\left\{p(t_1), p(t_2),\dots p(t_k)\right\}$, où $p(t_i)$ représente la priorité de t_i. Cette technique limite la durée des inversions de priorité mais n'interdit pas les interblocages.

Test d'ordonnançabilité

En considérant l'algorithme RM, la prise en compte du partage de ressources entraîne des modifications par rapport au test d'ordonnançabilité de cet algorithme (paragraphe 2.1.1) et on ne peut plus trouver de condition nécessaire et suffisante. Ce test peut être étendu de deux façons:

- La première méthode consiste à vérifier que chaque tâche peut respecter son échéance même si elle est bloquée par des tâches de priorité plus faible. Dans ce cas, l'ordonnancement d'un ensemble de n tâches périodiques $t=\left\{t_1,\dots t_i,\dots t_n\right\}$ en utilisant la technique à priorité héritée, est faisable si:

$$\forall i,\ 1\le i\le n,\ \sum_{j=1}^{i} C_j/P_j + B_i/P_i \le i\cdot\left(\sqrt[i]{2}-1\right) \qquad (2.6)$$

  où B_i est la durée maximale de blocage de la tâche t_i par les tâches de priorité inférieure ($t_{i+1},\dots t_n$). Le calcul de B_i est présenté dans [SRL, 90].

- La seconde méthode est plus simple à mettre en oeuvre, mais plus restrictive. Dans ce cas, l'ordonnancement d'un ensemble de n tâches périodiques $t=\left\{t_1,\dots t_i,\dots t_n\right\}$ en utilisant la technique à priorité héritée, est faisable si:

$$\sum_{i=1}^{n} C_i/P_i + \max_{1\le i\le n}\left\{B_i/P_i\right\} \le n\cdot\left(\sqrt[n]{2}-1\right) \qquad (2.7)$$

  Le calcul de B_i est le même que précédemment.

2. La priorité plafond

Principe

La technique à priorité héritée ne limite pas suffisamment la durée des inversions de priorité (toute tâche peut être bloquée par toutes les tâches de priorité inférieure et par le verrouillage de tous les sémaphores auxquels elle accède) et n'empêche pas les interblocages. C'est pourquoi la technique à priorité plafond a été définie: dans cette technique, une tâche, pour entrer en section critique, doit avoir une priorité strictement supérieure aux "valeurs plafonds" des sémaphores verrouillés par les autre tâches. Une "valeur plafond" est la "valeur maximale" parmi les priorités des tâches qui pourraient utiliser la ressource.

En conséquence, une tâche t peut préempter une tâche t' en section critique si et seulement si sa priorité est supérieure aux "valeurs plafonds" (ce qui veut dire que cette tâche t accède à des ressources auxquelles la tâche t' n'accède pas et que sa priorité est supérieure aux valeurs plafonds des sémaphores verrouillés par la tâche t').

Test d'ordonnançabilité

- En considérant l'algorithme RM, les tests présentés pour la technique à priorité héritée peuvent être considérés (le calcul de B_i peut être affiné [Mar, 94]);
- En considérant l'algorithme ED, une condition suffisante d'ordonnançabilité est la suivante [Bak, 91]:

$$\forall_{\substack{k \\ 1 \le k \le n}} \left(\sum_{i=1}^{n} \frac{C_i}{P_i} \right) + \frac{B_k}{P_k} \le 1 \qquad (2.8)$$

3. Algorithmes non-préemptifs: l'algorithme ED non-préemptif

Peu de travaux ont été faits sur les algorithmes non-préemptifs. En effet, dans un contexte monoprocesseur, même si un ordonnancement non-préemptif est plus simple à mettre en oeuvre qu'un ordonnancement préemptif (la préemption induit de fréquents changements de contexte d'exécution), on n'atteint pas cependant les mêmes taux d'ordonnançabilité à cause des inversions de priorités.

Un travail intéressant sur l'algorithme ED non-préemptif a cependant été fait. C'est le résumé de ce travail que nous présentons maintenant.

Certains chercheurs [JSM, 91] ont abordé le problème de l'ordonnancement non-préemptif de tâches périodiques et apériodiques à contraintes strictes. Ils ont tout d'abord, d'une part, défini des conditions nécessaires d'ordonnançabilité des tâches périodiques et, d'autre part, montré que ces conditions nécessaires suffisent également à l'ordonnançabilité de tâches apériodiques à contraintes strictes. Ils ont enfin démontré que si une tâche ne respecte pas son échéance, une des conditions nécessaires d'ordonnançabilité a été violée.

Nous présentons maintenant les résultats d'une analyse qui permet l'obtention des conditions nécessaires d'ordonnançabilité définies dans [JSM, 91]. Considérons un ensemble de tâches périodiques $t = \{t_1, \ldots t_i, \ldots t_n\}$, avec $t_i = \{C_i, P_i\}$ et $P_i \ge P_j$ si $i > j$, à échéance sur requête, et supposons un temps discret pour l'occurrence des requêtes des tâches.

L'obtention des conditions d'ordonnançabilité est basée:

- d'une part, sur la définition du pire cas de déphasage d'une tâche t_i ;

- d'autre part, dans l'hypothèse de ce pire cas de déphasage d'une tâche t_i, sur l'évaluation de la charge de travail du processeur.

3.1. Pire cas de déphasage d'une tâche t_i

Définition: On est dans le pire cas de déphasage d'une tâche t_i (figure 2.3) lorsque la requête de la tâche t_i se produit une unité de temps avant l'arrivée des requêtes des tâches plus prioritaires ($t_1,\dots t_{i-1}$).

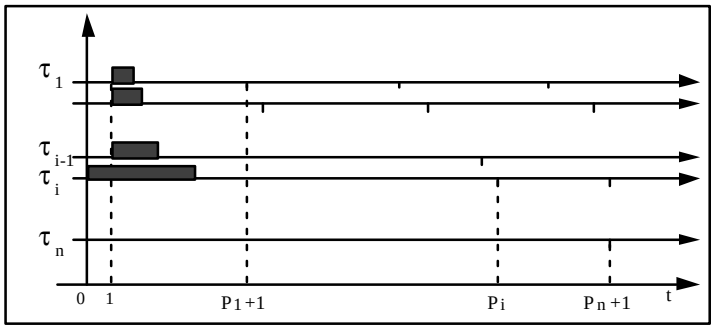

Figure 2.3: Pire cas de déphasage (tâche t_i)

Le pire cas de déphasage de la tâche t_i induit donc une inversion de priorité (t_i est traité avant les tâches $t_1,\dots t_{i-1}$). Les tâches $t_1,\dots t_{i-1}$ subissent donc un retard et ne pourront être considérées qu'à la fin de l'exécution de la tâche t_i. Il est donc essentiel de vérifier dans quelles conditions toutes les tâches $t_1,\dots t_{i-1}$ de priorités supérieures à t_i pourront satisfaire leurs échéances ($P_1,\dots P_{i-1}$). Pour ce faire, il faut, sur l'intervalle de temps, que nous appelons *intervalle de temps critique lié à la tâche* t_i (cet intervalle débute à l'invocation de la tâche t_i (instant 0) et finit avant l'échéance de t_i (instant P_i)), évaluer l'utilisation minimale demandée au processeur, compte tenu que toutes les invocations des tâches t_j de priorité supérieure à t_i (c'est-à-dire, qui ont des périodes inférieures) doivent avoir leurs échéances respectées. Nous disons utilisation minimale parce que les tâches t_k de priorité inférieure à t_i, c'est-à-dire qui ont des périodes supérieures, ont leur échéance après l'instant P_i et donc il n'est pas nécessaire qu'elles utilisent le processeur avant l'instant P_i.

L'intervalle de temps critique associé à la tâche t_i est noté: $[0, L]$, avec $P_1 + 1 \leq L < P_i$, soit encore $P_1 < L < P_i$.

3.2. Évaluation de l'utilisation minimale demandé au processeur sur l'intervalle de temps critique associé à la tâche t_i

Appelons $\{d_{0,L}\}_i$ cette utilisation minimale. On a:

$$\{d_{0,L}\}_i = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{P_j} \right\rfloor C_j \quad , P_1 < L < P_i \tag{2.9}$$

Cette expression comprend:

- d'une part, le temps d'utilisation relatif à l'exécution de la tâche t_i : C_i;
- d'autre part, la somme des temps d'utilisations des différentes tâches t_j ($j < i$) plus prioritaires que la tâche t_i; le facteur $\lfloor (L-1)/P_j \rfloor$ exprime le fait que, pour toute tâche t_j, seules les invocations qui débutent et finissent dans l'intervalle $L-1$ doivent être considérées. Les tâches t_k ($k > i$) moins prioritaires que la tâche t_i ne sont pas considérées puisque leurs échéances sont postérieures à l'instant P_i.

3.3. Test d'ordonnançabilité

L'ordonnancement de l'ensemble de n tâches: $t = \{t_1, \ldots t_i, \ldots t_n\}$ en utilisant un algorithme ED non-préemptif est faisable si:

- $\sum_{i=1}^{n} C_i / P_i \leq 1$ (2.10)

- $C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{P_j} \right\rfloor C_j \leq L \quad , \begin{cases} 1 < i \leq n \\ P_1 < L < P_i \end{cases}$ (2.11)

La première condition, équivalente à celle de l'algorithme ED, représente une garantie de non surcharge du système. Informellement, nous pouvons dire que la somme cumulative de la charge induite par chaque tâche, ne doit pas dépasser l'unité. La deuxième condition impose que la demande faite au processeur dans l'intervalle de temps critique $[0, L]$ associé à une tâche t_i $(\forall_i)$ doit être toujours inférieure à la longueur de cet intervalle.

3.3. Intérêt des algorithmes non-préemptifs

Sachant que l'exécution des tâches n'est jamais interrompue, l'implémentation d'un ordonnanceur non-préemptif est toujours plus simple que celle d'un ordonnanceur préemptif, car les changements de contextes ne se produisent qu'à la fin d'exécution des tâches (prévisibilité de fonctionnement). En plus, tant qu'on reste dans le cas monoprocesseur, on n'a plus besoin de mécanismes d'exclusion mutuelle (rendez-vous, sémaphores, …) pour protéger des données ou des ressources partagées. En conséquence, la charge d'exécution due au fonctionnement d'un ordonnanceur non-préemptif est généralement plus réduite.

Cependant, il faut noter que la préemption donne plus de liberté à l'ordonnanceur pour choisir une solution d'ordonnancement [CM, 94].

4. Contributions

Nous présentons maintenant des travaux que nous avons réalisés sur les algorithmes non-préemptifs et qui constituent, à notre avis, des contributions intéressantes dans le domaine des algorithmes non-préemptifs. Le travail le plus important est relatif à l'algorithme ED non-préemptif. Une réflexion sur l'algorithme RM non-préemptif est également présentée.

4.1. Algorithme ED non-préemptif

4.1.1. Justification et présentation de l'étude effectuée

Le travail sur l'algorithme ED non-préemptif effectué dans [JSM, 91] est intéressant, car c'est la première fois que des conditions nécéssaires et suffisantes d'ordonnançabilité pour un ordonnancement en-ligne non-préemptif (c.f. II.3) ont été énoncées. Cependant, la deuxième condition d'ordonnançabilité (condition spécifique de l'attribut "non-préemptif") énoncée est exprimée dans une échelle temporelle discrète, ce qui:

a) implique, d'une part, que toutes les périodes et durées des tâches soient des multiples de l'unité de temps choisie et, d'autre part, que le pire cas de déphasage soit égal à cette unité de temps,

b) induit une complexité de calcul incompatible avec l'attribut en-ligne de l'ordonnancement (car pour chaque tâche t_i, on doit évaluer l'utilisation minimale

$\{d_{0,L}\}_i$ demandée au processeur pour tous les L appartenant à l'ensemble $\{(P_1+1),(P_1+2),\ldots,(P_i-1)\}$).

Ce sont précisément ces deux constatations a) et b) qui ont motivé notre travail sur l'algorithme ED non-préemptif et qui nous ont donc amené:

- à considérer une échelle temporelle continue (les périodes et durées des tâches peuvent donc être quelconques et le déphasage initial peut être infiniment petit; la complexité des calculs est moins grande)
- et donc à reformuler la deuxième condition d'ordonnançabilité, dans le cadre de cette échelle temporelle continue, et à proposer une méthodologie d'évaluation de cette deuxième condition d'ordonnançabilité facile à mettre en oeuvre (et par conséquent, compatible avec l'attribut "en-ligne" de l'ordonnancement).

4.1.2. Considération d'une échelle temporelle continue

1. Expression de la deuxième condition d'ordonnançabilité

Reprenons l'expression de l'utilisation minimale demandé au processeur pendant l'intervalle de temps critique lié à une tâche i $\{d_{0,L}\}_i$ et remplaçons la variable temporelle discrète L et l'intervalle de temps unitaire par, respectivement, les variables temporelles continues t et a .

Le pire cas de déphasage de la tâche i appartenant à l'ensemble de tâches $T=\{t_1, t_2,\ldots t_i,\ldots t_n\}$ est obtenu lorsque $(a \to 0^+)$ (figure 2.4).

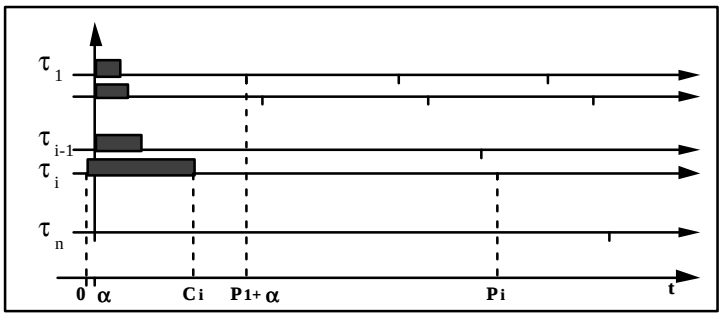

Figure 2.4: Pire cas de déphasage

L'utilisation minimale demandée au processeur pendant l'intervalle de temps critique lié à la tâche t_i (intervalle $[0,t]$ avec $P_1 + a \le t < P_i$ et $(a \to 0^+)$) est maintenant notée $\{d_{0,t}\}_i$ et s'exprime de la manière suivante:

$$\{d_{0,t}\}_i = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{t-a}{P_j} \right\rfloor C_j \qquad (2.12)$$

La deuxième condition d'ordonnançabilité s'exprime maintenant sous la forme suivante:

$$\{d_{0,t}\}_i \le t \quad , \quad \forall_{P_1+a \le t < P_i} \ , \ \forall i_{1<i\le n} . \qquad (2.13)$$

L'évaluation de cette deuxième condition d'ordonnançabilité nécessite:

- tout d'abord de calculer, $\forall i,\ 1 < i \le n$, la fonction $\{d_{0,t}\}_i$ qui est une fonction en escalier croissante,

- dont les discontinuités se produisent à tous les instants t, tels que $(t-a)$ est un multiple de P_j ($\forall j,\ 1 \le j \le i-1$), c'est-à-dire, $t - a = k \cdot P_j$, soit $t = k \cdot P_j + a$ ou encore comme $\left(a \to 0^+\right)$, $t = k \cdot P_j^+$; ces instants représentent les instants d'arrivées des requêtes des tâches t_j;
- dont la valeur sur le palier à l'instant $t = k \cdot P_j^+$ est la durée de l'utilisation demandée au processeur sur l'intervalle $\left[0, k \cdot P_j^+\right]$; nous appelons $u_i\left(\left[0, k \cdot P_j^+\right]\right)$ cette durée,

- et ensuite, sachant que $\left\{d_{0,t}\right\}_i$ est une fonction en escalier croissante, il suffit de vérifier qu'à chaque instant $t = k \cdot P_j^+$, $\forall i,\ 1 < i \le n$, la durée de l'utilisation demandée au processeur $u_i\left(\left[0, k \cdot P_j^+\right]\right)$ est strictement inférieure à $k \cdot P_j^+$.

2. Reformulation de la deuxième condition d'ordonnançabilité

Nous introduisons la notion de demande normalisé $W_i(t) = \dfrac{\left\{d_{0,t}\right\}_i}{t}$, ce qui nous amène à reformuler la deuxième condition d'ordonnançabilité sous la forme suivante:

$$W_i(t) \le 1 \qquad \forall t:\ P_1 \le t < P_i\ ,\ \forall i:\ 1 < i \le n\ . \tag{2.14}$$

Notons que comme $\left\{d_{0,t}\right\}_i$ est une fonction en escalier croissante, dont les discontinuités se produisent aux instants $k \cdot P_j^+$ (fonction ouverte à gauche et fermée à droite des instants de discontinuité), la demande normalisée est une fonction qui présente des maxima aux instants $k \cdot P_j^+$ et qui est décroissante entre les instants $k \cdot P_j^+$.

La vérification de la deuxième condition d'ordonnançabilité demande à vérifier, $\forall i,\ 1 < i \le n$, qu'à chaque instant $k \cdot P_j^+$, la demande normalisée $W_i\left(k \cdot P_j^+\right)$ qui est définie par le rapport $\dfrac{u_i\left(\left[0, k \cdot P_j^+\right]\right)}{k \cdot P_j^+}$ est strictement inférieure à 1. La notion "strictement inférieure à 1" découle de la définition du pire cas de déphasage (a toujours positif).

Il nous est paru intéressant d'introduire cette valeur de demande normalisée qui permet de traiter sous la forme d'un rapport, c'est-à-dire de manière (à notre avis) plus parlante, la deuxième condition d'ordonnançabilité.

4.1.3. Méthode d'évaluation de la deuxième condition d'ordonnançabilité

La présentation précédente a montrée l'importance des instants $t = k \cdot P_j^+$ pour évaluer la deuxième condition d'ordonnançabilité. Nous appelons ces instants $t = k \cdot P_j^+$ les points d'ordonnancement.

La méthode proposée est la suivante:

1- pour tout $i,\ 1 < i \le n$:

a) définir l'ensemble S_i des points d'ordonnancement:

$$S_i = \bigcup_{j=1}^{i-1} \left\{ \bigcup_{k=1}^{\lceil P_i / P_j - 1 \rceil} \left(k \cdot P_j^+\right) \right\} \tag{2.15}$$

b) calculer les maxima de la demande normalisée $W_i(t)$, c'est-à-dire on calcule la valeur de la demande normalisée en chaque point d'ordonnancement:

on a: $\forall_i$, $\forall_{k.P_j^+ \in S_i}$, $W_i\left(k \cdot P_j^+\right) = \dfrac{u_i\left(\left[0, k \cdot P_j^+\right[\right)}{k \cdot P_j^+} = \dfrac{C_i + \sum_{l=1}^{i-1} \left\lceil \left(k \cdot P_j^+\right) / P_l \right\rceil C_l}{k \cdot P_j^+}$ (2.16)

Appelons W_i l'ensemble de ces valeurs.

c) déterminer la valeur maximale sur l'ensemble des valeurs $W_i\left(k \cdot P_j^+\right)$; appelons $W_{i_{max}}$ cette valeur maximale:

$$W_{i_{max}} = max\ W_i\left(k \cdot P_j^+\right),\ \ \forall_{k \cdot P_j^+ \in S_i} \tag{2.17}$$

2- la phase précédente permet d'obtenir l'ensemble des valeurs W_i relatives à chaque tâche i $(1 < i \le n)$. On détermine la valeur maximale de ces valeurs $W_{i_{max}}$, que l'on appelle W :

$$W = \max_{1<i\le n} \left\{W_{i_{max}}\right\} \tag{2.18}$$

3- la deuxième condition d'ordonnançabilité est vérifiée si $W < 1$

4.1.4. Exemple d'application

Considérons l'ensemble de tâches $T = \left\{t_1, t_2, t_3, t_4\right\}$, tel que: $\left\{\left(C_i, P_i\right)\right\} = \left\{(2,6), (3,10), (4,20), (4,30)\right\}$, pour lequel la première condition d'ordonnançabilité est vérifiée $\left(\sum_{i=1}^{4} C_i / P_i = 0.9667 \le 1\right)$.

Appliquons la méthodologie que nous venons de présenter pour le calcul de la deuxième condition d'ordonnançabilité:

1- Pour tout $i,\ 2 \le i \le 4$:

a) les ensembles S_i des points d'ordonnancement sont:

$$S_2 = \left\{6^+\right\},\ S_3 = \left\{6^+, 10^+, 12^+, 18^+\right\},\ S_4 = \left\{6^+, 10^+, 12^+, 18^+, 20^+, 24^+\right\}$$

b) les ensembles W_i des maxima des demandes normalisées sont:

$$W_2 = \left\{\frac{5}{6^+}\right\},\ W_3 = \left\{\frac{6}{6^+}, \frac{9}{10^+}, \frac{11}{12^+}, \frac{13}{18^+}\right\},\ W_4 = \left\{\frac{6}{6^+}, \frac{9}{10^+}, \frac{11}{12^+}, \frac{13}{18^+}, \frac{20}{20^+}, \frac{22}{24^+}\right\};$$

c) la valeur maximale des demandes normalisées: $W_{i_{max}}$, est:

$$W_{2_{max}} = \frac{5}{6^+},\ W_{3_{max}} = \frac{6}{6^+} \text{ et } W_{4_{max}} = \frac{6}{6^+} \tag{2.19}$$

Nous utilisons encore la notation suivante: $W_{2_{max}} = 0.833^-$, $W_{3_{max}} = 1^-$ et $W_{4_{max}} = 1^-$ qui veut visualiser des relations de type "strictement inférieur".

2- on obtient: $W = \max_{1<i\le 4} \left\{W_{i_{max}}\right\} = 1^-$

3- Donc la deuxième condition d'ordonnançabilité est vérifiée ($W < 1$).

Nous représentons sur les figures 2.5 et 2.6 les séquences temporelles des tâches (en haut des figures) et les demandes normalisées (en bas des figures); les courbes relatives aux demandes normalisées visualisent à la fois les points d'ordonnancement et les maxima des demandes normalisées en ces points.

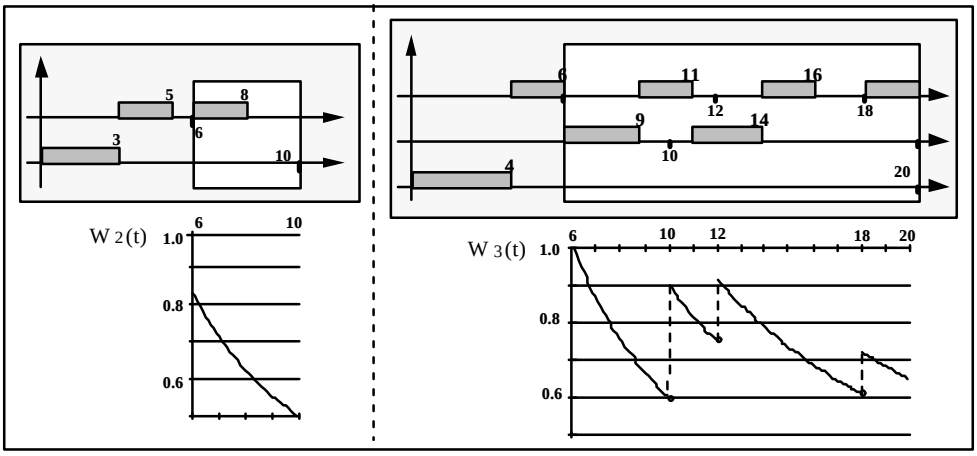

Figure 2.5: Séquences temporelles et demande normalisée $W_2(t)$ et $W_3(t)$

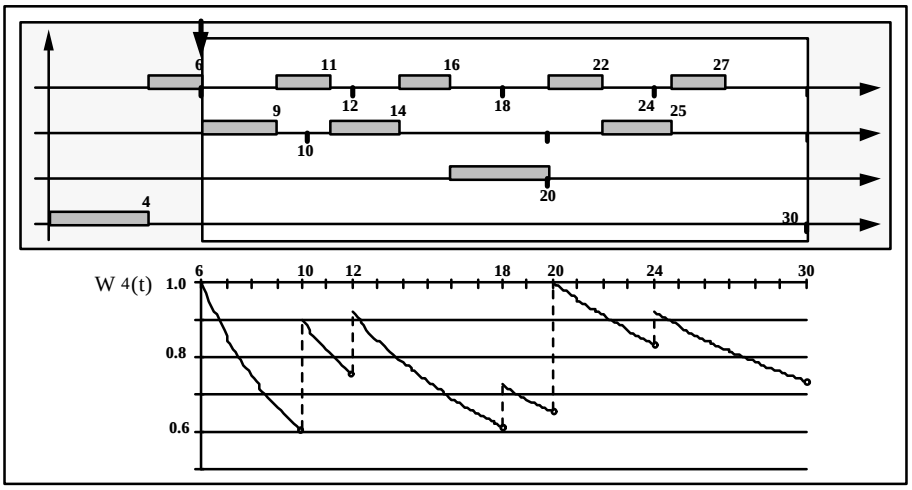

Figure 2.6: Séquence temporelle et demande normalisée $W_4(t)$

4.2. Algorithme RM non-préemptif

4.2.1. Proposition

A notre connaissance, il n'y a jamais eu de travaux sur les algorithmes RM non-préemptifs. Cependant, en utilisant les résultats de l'étude sur le partage de ressources avec les algorithmes préemptifs, et en particulier, sur les techniques à priorité héritée [SRL, 90], on peut définir une condition suffisante d'ordonnançabilité pour un algorithme RM non-préemptif.

Considérons les deux expressions présentées au paragraphe 2.1.4 et qui représentent deux tests d'ordonnançabilité de l'algorithme RM préemptif:

- premier test: $$\forall i,\ 1 \le i \le n,\ \sum_{j=1}^{i} \frac{C_j}{P_j} + \frac{B_i}{P_i} \le i \cdot \left(\sqrt[i]{2} - 1\right) \qquad (2.30)$$

  où B_i représente la durée maximale du blocage d'un travail de la tâche t_i par un travail d'une tâche de priorité inférieure $t_{I+1}, \ldots t_n$.

- deuxième test: $$\forall i,\ 1 \le i \le n,\ \sum_{i=1}^{n} \frac{C_i}{P_i} + \max_{1<i\le n} \left\{ \frac{B_i}{P_i} \right\} \le n \cdot \left(\sqrt[n]{2} - 1\right) \qquad (2.31)$$

  où B_i a la même signification que précédemment.

Ces deux formules peuvent être utilisées pour définir deux tests d'ordonnançabilité pour l'algorithme RM non-préemptif, en considérant que le travail de la tâche t_i est maintenant bloqué pendant "toute la durée d'exécution" d'un travail d'une tâche de priorité inférieure $t_{I+1}, \ldots t_n$ (attribut de non-préemption).

Nous définissons donc B_i comme étant la durée maximale des tâches de priorité inférieure à la tâche t_i, c'est-à-dire:

$$B_i = \max_{i+1 \le j \le n} \{C_j\} \tag{2.32}$$

Notons que les deux tests sont pessimistes (comme le test de l'algorithme RM préemptif) comme nous le montrons sur l'exemple suivant.

4.2.2. Exemple

Nous considérons un exemple où nous appliquons, à la fois, le premier et le deuxième tests d'ordonnançabilité.

Soit l'ensemble de tâches $t = \{t_1, t_2, t_3\}$ avec $\{C_i, P_i\} = \{(2,8), (2,10), (5,20)\}$, ordonnancé par l'algorithme RM non-préemptif.

Nous représentons sur les figures 2.8a et 2.8b l'intervalle de temps critique associés, respectivement, à la tâche t_2 et à la tâche t_3. On voit donc que l'ordonnancement est possible quel que soit le pire cas de déphasage considéré.

Cependant, les tests d'ordonnançabilité ne sont pas satisfaits quand on vérifie l'ordonnançabilité de t_3, ce qui montre que ces tests d'ordonnançabilité sont pessimiste.

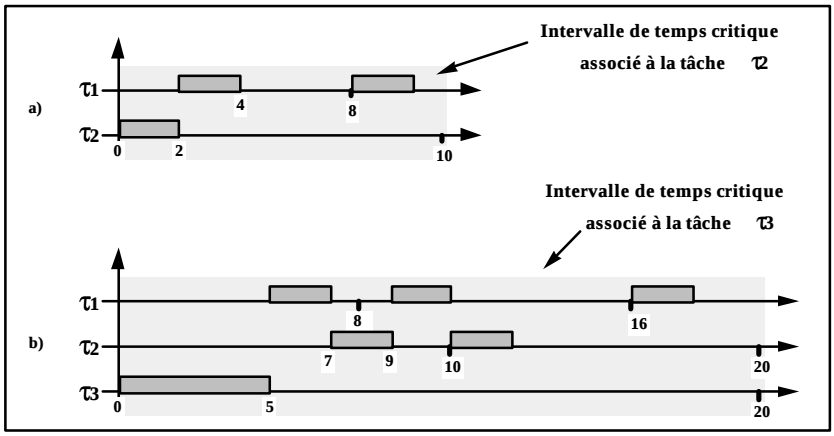

Figure 2.8:

5. Conclusion

Outre la présentation des principaux algorithmes pour des tâches indépendantes contraintes temporellement (algorithmes RM, DM, ED) et de la problématique du partage de ressources (techniques par héritage), nous voulons insister sur le travail présenté relativement aux algorithmes non-préemptifs (l'attribut "non-préemptif" est important pour l'ordonnancement du transfert de messages):

- algorithme ED non-préemptif (deuxième condition d'ordonnançabilité [JSM, 91], qui est la condition spécifique de l'attribut "non-préemptif" et qui a été exprimée dans une échelle temporelle discrète): nous avons, d'une part, développé une extension de cette deuxième condition à une échelle temporelle continue (ce qui permet de considérer des

périodes et des durées quelconques pour les tâches, ainsi que des déphasages initiaux infiniment petits), et, d'autre part, proposé une méthode de calcul relativement simple et facilement implantable en-ligne;

- algorithme RM non-préemptif: ne connaissant pas de travaux dans ce domaine, nous avons suggéré une condition d'ordonnançabilité qui exploite le résultat sur les techniques à priorité héritée pour le partage de ressources.

6. Références

[AB, 90] N. Audsley, A. Burns; Real-Time Systems Scheduling; YCS 134, Departement of Computer Science, University of York, 1990

[Bak, 91] T. Baker; Stack-Based Scheduling of Realtime Processes; Journal of Real-Time Systems, 3, pages 67-99, 1991

[CC, 89] H. Chetto, M. Chetto; Some Results of the Earliest Deadline Scheduling Algorithm; IEEE Tr. on Software Engineering, 15(10), pages 1261-1269, 1989

[CM, 94] C. Cardeira, Z. Mammeri; Ordonnancement de Tâches dans les Systèmes Temps Réel et Répartis: Algorithmes et Critères de Classification; APII 28(4), pp 353-384, 1994

[Del, 94] J. Delacroix; Un contrôleur d'ordonnancement temps-réel pour la stabilité de earliest deadline en surcharge: le régisseur; Thèse de Doctorat, CNAM, Paris, 1994.

[JP, 86] M. Joseph, P. Pandaya; Finding Response Times in a Real-Time System; The Computer Journal, 29(5), pp. 390-395, 1986.

[JSM, 91] K. Jeffay, D. Stanat, C. Martel; On Non-Preemptive Scheduling of Periodic and Sporadic Tasks, in Proc. of RTSS'91 - IEEE Real-Time Systems Symposium, San Antonio, Texas , December 1991

[LL, 73] C. Liu and J. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, Journal of ACM, vol. 29, nº 1, 1973, pp 46-61

[LM, 80] J. Leung, M. Merril; A Note on Preemptive Scheduling of Periodic Real-Time Tasks; Information Processing Letters, 11(3), pp 115-118, November 1980.

[LSD, 89] J. Lehoczky, L. Sha, Y. Ding; The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour; IEEE RTSS'89, pages 166-171, December 1989

[LSS, 87] J. Lehoczky, L. Sha, J. Strosnider; Enhanced Aperiodic Responsiveness in Hard Real-Time Environments; IEEE RTSS'87, pages 261-270, 1987

[Mar, 94] P. Martineau; Ordonnancement en-ligne dans les systèmes informatiques temps-réel; Thèse de Doctorat (nº ED 82-93), Ecole Centrale de Nantes, 21 Octobre 1994.

[SRL, 90] L. Sha, R. Rajkumar, J. Lehoczky; Priority Inheritance Protocols: An Approach to Real-Time Synchronization; IEEE Tr. on Computers, 39(9), September 1990

[SSL, 89] B. Sprunt, J. Lehoczky, L. Sha; Aperiodic Task Scheduling for Hard Real-Time Systems; Journal of Real-Time Systems, 1, pages 27-60, 1989