Cyclicité des séquences d'ordonnancement au plus tôt des systèmes de tâches temps réel à contraintes strictes

Emmanuel GROLLEAU, Annie CHOQUET–GENIET, Francis COTTET
LISI–ENSMA
Site du FUTUROSCOPE BP 109
86960 FUTUROSCOPE Cedex
E–mail grolleau@ensma.fr, ageniet@ensma.fr, cottet@ensma.fr

Résumé : Nous nous intéressons aux ordonnancements au plus tôt de systèmes de tâches temps réel à contraintes strictes qui peuvent partager des ressources et se synchroniser. Lorsque la validation est obtenue par simulation, la durée de simulation requise doit être connue. De plus la construction d'une séquence potentiellement infinie n'est possible que si la séquence contient un cycle. Nous montrons que le temps de simulation nécessaire est connue pour tout système de tâches, et nous affinons les résultats déjà connus [7] puis nous montrons que le système a un comportement cyclique.
Mots clés : temps réel, ordonnancement, en ligne, hors ligne, simulation, cyclicité.

1. Introduction

Un système temps réel se distingue des autres systèmes informatiques par le fait qu'il est soumis à des contraintes temporelles fortes, dues à la criticité de certaines actions, typiquement des actions qui permettent au système d'interagir avec l'environnement. Une application temps réel peut être vue comme un système de tâches se synchronisant et partageant des ressources telles que le processeur dans les systèmes monoprocesseurs, les processeurs dans les systèmes répartis, ou la sortie vers un terminal de contrôle [10] dans le but de contrôler un procédé.

Nous nous plaçons ici dans un contexte monoprocesseur, et nous supposons que les tâches composant le système sont toutes périodiques. Nous nous intéressons à la correction du système. Dans le cadre temps réel, on entend par correction du système non seulement la correction algorithmique classique, avec en particulier l'assurance qu'aucun blocage n'interviendra, mais aussi la correction temporelle : chaque tâche doit s'exécuter à l'intérieur d'une plage de temps dont la taille dépend des caractéristiques physiques du procédé. L'étude de la correction temporelle du système passe par l'étude des ordonnancements possibles. On peut utiliser :

- Soit des techniques d'ordonnancement en ligne : de nombreuses études ont été menées dans ce domaine, concernant des algorithmes d'ordonnancement basés sur les valeurs des paramètres temporels caractérisant les tâches (*Earliest Deadline* [7], *Rate Monotonic* [8]) avec ou sans protocole de gestion de ressources.
- Soit des techniques hors ligne, ce qui revient à construire un séquenceur. Le choix de la séquence utilisée dépend de critères fixés par le client, critères qui peuvent ne pas être exprimables par les seuls paramètres temporels, comme par exemple des critères d'importance des tâches [6].

Dans le premier cas, il nous faut montrer que l'algorithme choisi fournit une séquence qui sera temporellement correcte. Il ne faut pas perdre de vue que le système de contrôle doit fonctionner aussi longtemps que le procédé existe, et qu'il faut donc établir l'existence d'un régime permanent correct. Sous des hypothèses restrictives (tâches indépendantes, tâche avec sections critiques de durée unitaires) [3][8][11], on dispose de conditions simples qui peuvent permettre de conclure. Mais dans le cas général de tâches interagissantes, seule la simulation permet de conclure. Ce qui amène à la question primordiale : quelle est la durée minimale de simulation qui permet de conclure ?

Dans le deuxième cas, il n'est pas envisageable de construire pas à pas une séquence infinie. Il faut donc trouver deux morceaux de séquences S et M tels que la séquence SM^* soit correcte. Ceci suppose qu'après une montée en charge, le système aura un comportement cyclique.

Le but de cet article est d'établir l'existence de ce comportement cyclique, et de montrer que l'on sait borner la durée de simulation nécessaire pour établir la validité d'un système, et, le cas échéant, pour construire S et M.

Dans le cas où toutes les tâches sont synchrones au démarrage (toutes les tâches sont activées pour la première fois au même instant), la durée de simulation nécessaire est égale au plus petit commun multiple (PPCM) des périodes, noté P. En effet à l'instant 0 toutes les tâches sont activées et à l'instant P, toutes les tâches doivent avoir terminé leur exécution et sont réactivées. Le système se trouve donc globalement dans le même état qu'au début (mêmes compteurs ordinaux et mêmes horloges locales) et l'ordonnancement est donné par la séquence S de 0 à P répétée à l'infini, que nous notons S^*. Il a été également montré dans [7] que, si l'on ordonnance un système de tâches indépendantes non synchrones au démarrage par ED, la durée de simulation est égale au maximum des dates de réveil des différentes tâches, noté r, plus deux fois P, et le système est dans le même état au temps r+P qu'au temps r+2P.

Nous étudions ici les systèmes de tâches interagissantes, en supposant simplement que l'on utilise un ordonnancement au plus tôt : le système n'est inactif que si aucune tâche ne peut s'exécuter.

L'article est organisé de la façon suivante : nous présentons tout d'abord le modèle temporel utilisé, ainsi que les hypothèses structurelles faites. Nous montrons ensuite que si le système a une charge de 100%, le nombre de temps creux (temps d'inactivité) est borné, puis nous montrons que la date d'occurrence du dernier temps creux est bornée par r+P. Enfin nous établissons la cyclicité des ordonnancements au plus tôt.

2. Définitions et notations

2.1 Modèle temporel

Un système de tâches temps réel est composé d'un ensemble de tâches cycliques $\tau_{i,\ i=1..n}$ caractérisées par quatre paramètres temporels [8] (figure 1) :

- r_i la date de première activation
- C_i la charge, durée de traitement nécessaire à chaque exécution
- R_i le délai critique, temps donné à la tâche pour se terminer après chacune de ses activations
- T_i période d'activation

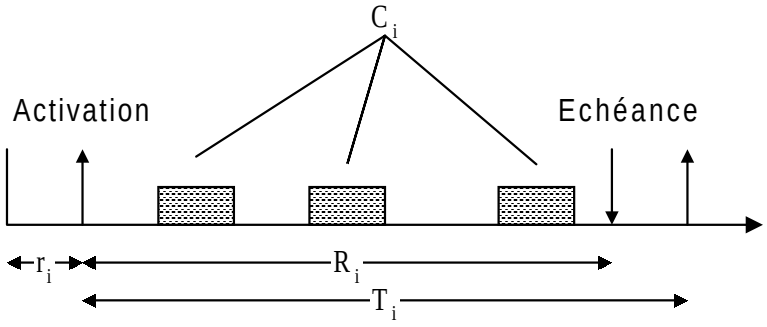

Figure 1 : Paramètres temporels des tâches

Une tâche est donc caractérisée temporellement par $\tau_i = <r_i, C_i, R_i, T_i>$. Nous appelons $P=PPCM_{i=1..n}(T_i)$ la métapériode du système, et $r = max_{i=1..n}(r_i)$.

2.2 Hypothèses structurelles

Nous adoptons les hypothèses suivantes :

- Les tâches sporadiques sont prises en compte par des serveurs périodiques. En effet rappelons que nous nous intéressons à la validation hors–ligne et que celle–ci n'a de sens que si toutes les occurrences des tâches sont connues à l'avance.
- Les tâches peuvent se synchroniser et partager des ressources. Nous utilisons indifféremment les termes messages et synchronisations qui expriment des contraintes de précédence entre certaines parties de corps de tâches.
- Les messages asynchrones, qui n'expriment pas de précédence entre des parties de tâches peuvent être considérés des ressources partagées de type « boîte aux lettres » et seront donc traités à ce titre.
- Lorsque deux tâches se synchronisent, les fréquences de la tâche émettrice et de la tâche réceptrice doivent être corrélées. En effet si deux tâches τ_i et τ_j se synchronisent, et si τ_i attend $a_{i,j}$ (pour attente) messages de τ_j pendant chacune de ses exécutions et que τ_j envoie $e_{j,i}$ (pour émission) messages à destination de τ_i pendant chacune de ses exécutions, alors une condition nécessaire (CN) d'ordonnançabilité est $\frac{e_{j,i}}{T_j} \geq \frac{a_{i,j}}{T_i}$. Cette CN traduit le fait que la fréquence d'émission doit être au moins égale à la fréquence de réception. Si la fréquence d'émission est supérieure à la fréquence de réception, $\frac{e_{j,i}}{T_j} = \frac{a_{i,j}}{T_i} + b$ avec b>0, $b \times PPCM_{i=1..n}(T_i)$ messages émis ne seront pas lus au cours de chaque métapériode. Dans ce cas il devient délicat de parler de synchronisation. Nous nous limiterons donc aux systèmes de tâches dont les émissions ont même fréquence que les réceptions i.e. $\frac{e_{j,i}}{T_j} = \frac{a_{i,j}}{T_i}$ que nous appelons condition nécessaire de synchronisme **(CN 1)**.
- Une tâche ne peut pas attendre de message de synchronisation à l'intérieur d'une section critique. En effet il est préférable qu'une tâche sorte le plus vite possible de sa section critique.

Ce sont les seules contraintes sur la structure des tâches étudiées, et toutes les imbrications de prises et libérations de ressources, et les encapsulations d'envois de synchronisation à l'intérieur des sections critiques sont permises.

2.3 Corrélation entre tâches synchronisées

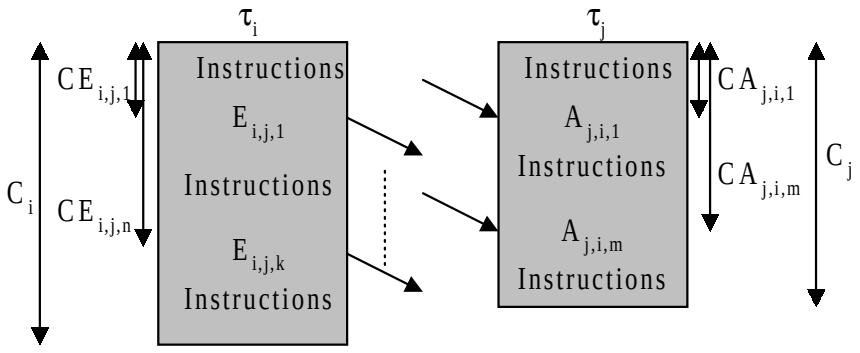

Figure 2 : Tâches se synchronisant

2.3.1 Point de vue relatif

Nous considérons le corps d'une tâche τ_i, et à l'intérieur de ce corps la $k^{ième}$ ($k \in \{1, ..., e_{i,j}\}$) émission vers τ_j. Nous notons $E_{i,j,k}$ cette émission et $CE_{i,j,k}$ la durée d'exécution de la tâche τ_i avant $E_{i,j,k}$. Notons que $E_{i,j,k}$ et $CE_{i,j,k}$ sont des paramètres intrinsèques de la tâche τ_i (figure 2).

De la même manière nous notons $A_{i,j,k}$ l'attente par τ_i du $k^{ième}$ message en provenance de τ_j et $CA_{i,j,k}$ la durée de la tâche τ_i avant $A_{i,j,k}$ ($k \in \{1, ..., a_{i,j}\}$). De façon similaire aux émissions, $A_{i,j,k}$ et $CA_{i,j,k}$ sont des paramètres constants de la tâche τ_i.

2.3.2 Point de vue absolu

Si on numérote les messages émis depuis l'instant 0, le numéro absolu d'une émission de message $E_{i,j,k}$ lors de l'activation en cours à un instant t est donné par Nombre_d'activations_de_τ_i*$e_{i,j}$+k, ce qui s'écrit : $e_{i,j}\left\lfloor \frac{t - r_i}{T_i} \right\rfloor + k = NE_{i,j}(t,k)$ où $\lfloor \ \rfloor$ est la partie entière inférieure.

De même le numéro absolu d'une attente de message $A_{i,j,k}$ pendant l'activation en cours à l'instant t est $a_{i,j}\left\lfloor \frac{t - r_i}{T_i} \right\rfloor + k = NA_{i,j}(t,k)$.

Réciproquement, si on considère la $p^{ième}$ attente absolue de message, elle correspond, pour l'activation en cours, à l'action $A_{i,j,k}$ avec $k=((p-1) \equiv a_{i,j})+1$, où $\equiv$ est le modulo. L'émission correspondante (la $p^{ième}$ en numérotation absolue) est l'émission $E_{j,i,M_{i,j}(k,t)}$ avec :

$M_{i,j}(k,t)=((NA_{i,j}(t,k)-1) \equiv e_{j,i})+1 = \left(\left(a_{i,j}\left\lfloor \frac{t - r_i}{T_i} \right\rfloor + k - 1\right) \equiv e_{j,i}\right) + 1$. En effet la CN1 oblige les fréquences d'émissions et de réceptions à être identiques. $\left\lfloor \frac{t - r_i}{T_i} \right\rfloor$ est le nombre de périodes terminées de τ_i au temps t, pendant chacune d'elles $a_{j,i}$ messages ont été attendus, et dans la période courante au temps t, $A_{i,j,k}$ est le $k^{ième}$ message attendu. Nous prenons le modulo $e_{i,j}$ auquel nous ajoutons 1 pour obtenir le numéro de l'émission qui est entre 1 et $e_{i,j}$. On peut remarquer d'une part que $M_{i,j}(k,t)$ n'est définie que pour $t \geq r_i$ et que cette fonction, de par la CN1, est périodique de période $PPCM(T_i,T_j)$ lorsque $t \geq r_i$.

2.4 Les temps creux

La charge totale d'un système de tâches est $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$, et lorsqu'elle est inférieure à 100% ($U<1$), il y a au moins $P(1-U)$ temps creux (temps pendant lequel le processeur ne fait rien) pendant chaque métapériode. Ces temps creux, qui se retrouvent toutes les métapériodes, sont qualifiés de temps creux cycliques. Les temps creux cycliques peuvent être modélisés par une tâche appelée tâche creuse $\tau=<0,P(1-U),P,P>$.

Lorsque les tâches ne sont pas synchrones au démarrage, il se peut qu'au début de l'ordonnancement, on trouve des temps creux, que nous qualifions d'acycliques. Il peut y en avoir par exemple lorsque toutes les tâches ne sont pas encore réveillées, avant r, mais il peut aussi y en avoir après la date r. Contrairement aux temps creux cycliques, ces temps creux ne se retrouveront pas à chaque métapériode, nous les qualifions donc de temps creux acycliques.

Afin de mieux différencier les temps creux cycliques des temps creux acycliques, nous utiliserons toujours une tâche oisive qui absorbera les temps creux cycliques. Ainsi nous

travaillons avec des systèmes de charge égale à 100% et tous les temps creux qui ont lieu sont des temps creux acycliques.

Pour exemple considérons le système de tâches indépendantes suivant :

$\tau_1 = <0,1,4,4>$, $\tau_2 = <1,3,6,6>$, $\tau_3 = <3,1,4,4>$ est ordonnancé par *Earliest Deadline* de la façon suivante :

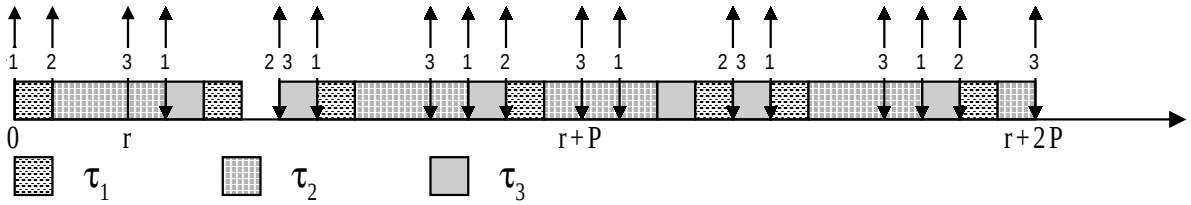

Figure 3 : Ordonnancement du système de tâches par Earliest Deadline

Bien que la charge de ce système soit de 100%, un temps creux apparaît au temps 6, c'est à dire après que toutes les tâches aient été activées. Ce temps creux est un temps creux acyclique, résidu des perturbations engendrées par la montée en charge du système.

3. Le nombre de temps creux acycliques est borné

Nous montrons dans ce paragraphe que le nombre de temps creux acycliques est borné.

Soit un système R composé de n tâches $\tau_i <r_i,C_i,R_i,T_i>$ tel que $\exists j \setminus r_j=0$ et $\exists k \setminus r_k>0$: nous nous plaçons dans le cas général de tâches non synchrones au démarrage. Etudions une tâche τ_i lors d'un comportement valide du système.

Soit k_i entier naturel, le nombre de périodes de τ_i écoulées avant r : $k_i = \left\lfloor \frac{r - r_i}{T_i} \right\rfloor$.

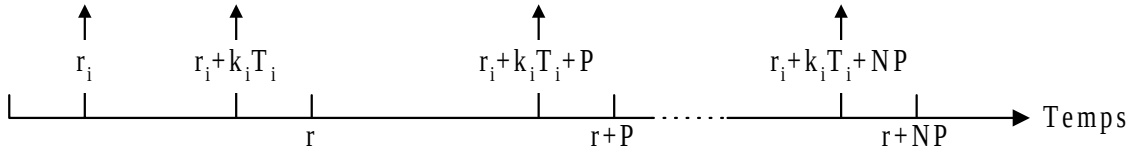

Figure 4 : Les activations de τ_i au cours du temps

La tâche τ_i a été exécutée au moins k_i fois avant r. Pour ces exécutions, le processeur a dû lui consacrer k_iC_i unités de temps. τ_i est activée pour la dernière fois avant r au temps $r_i+k_iT_i$. Elle est exécutée exactement $\frac{P}{T_i}$ fois entre $r_i+k_iT_i$ et $r_i+k_iT_i+P$. Le processeur devra donc lui consacrer exactement $k_iC_i + P\frac{C_i}{T_i}$ unités de temps entre 0 et $r_i+k_iT_i+P$. Pour $N \geq 0$, le processeur devra consacrer exactement $k_iC_i + NP\frac{C_i}{T_i}$ unités de temps à τ_i entre 0 et $r_i+k_iT_i+NP$.

Pour le système de tâches en entier, le processeur devra travailler au moins $\sum_{i=1}^{n}\left(k_iC_i + NP\frac{C_i}{T_i} \right)$ unités de temps entre 0 et r+NP, c'est à dire $NPU + \sum_{i=1}^{n} k_iC_i$ unités de temps.

Nous rappelons que U=1. Dans ce cas le processeur devra travailler au moins $NP + \sum_{i=1}^{n} \left\lfloor \frac{r - r_i}{T_i} \right\rfloor C_i$ entre 0 et r+NP et ceci pour tout N≥0. En régime permanent, le nombre de temps creux acycliques est donc borné par : $r - \sum_{i=1}^{n} \left\lfloor \frac{r - r_i}{T_i} \right\rfloor C_i$. On peut bien sûr affiner et réduire cette borne mais ce n'est pas ici notre propos.

Dans une séquence d'ordonnancement valide d'un système, de la forme MS^*, aucun temps creux acyclique ne doit apparaître dans la séquence S. Il est donc clair que le temps de simulation nécessaire pour valider une séquence d'ordonnancement doit être supérieur ou égal à la date d'occurrence du dernier temps creux acyclique plus P. D'où les deux questions primordiales : le dernier temps creux acyclique a–t–il une date d'occurrence limite et surtout le système de tâches se retrouve–t–il dans le même état P unités de temps après le dernier temps creux acyclique ?

4. Dernier temps creux acyclique pour les ordonnancements au plus tôt

4.1 Les chaînes de blocage

Nous considérons les séquences d'ordonnancement au plus tôt, c'est à dire telles qu'un temps creux n'a lieu que lorsqu'aucune tâche ne peut avancer (i.e. le système est globalement bloqué). Cela se produit dans l'un des deux cas suivants :

- Toutes les tâches ont terminé, nous parlons alors de temps creux acyclique d'inactivité.
- Seules des tâches en attente de message de synchronisation ne sont pas terminées. Nous parlons alors de temps creux acyclique de blocage.

Remarque : Si l'on ajoute au système une tâche oisive montant la charge processeur à 100%, la contrainte d'ordonnancer au plus tôt ne s'applique plus qu'aux temps creux acycliques, ce qui semble être le cas pour tout algorithme connu.

Il n'y a pas d'autre cause possible de blocage global du système si ce n'est l'occurrence d'un interblocage dû aux ressources ou bien une cyclicité du graphe de précédence des tâches, mais nous supposons que le système est correctement programmé (i.e. il n'y a pas d'interblocage). Lorsque le système est bloqué, si il y a une tâche non terminée, alors il y a au moins une tâche τ_i qui est en attente d'un message de synchronisation de la part d'une tâche non réactivée. Les tâches en attente d'un message de τ_i sont elles aussi bloquées et peuvent bloquer à leur tour d'autres tâches. Cela forme des chaînes de blocage (figure 5). Lorsque toutes les tâches actives sont dans une chaîne de blocage, il y a un temps creux acyclique de blocage.

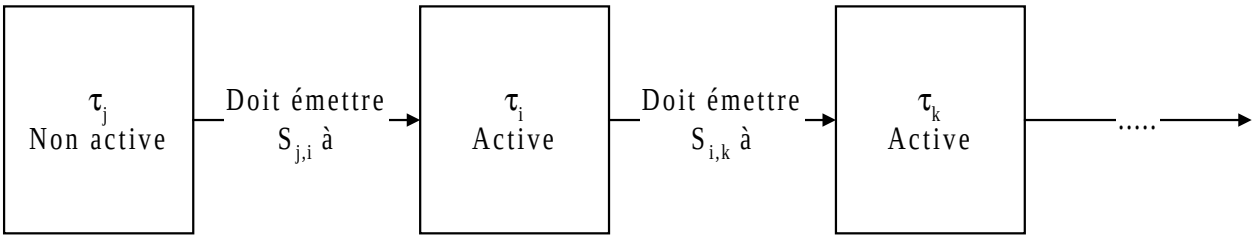

Figure 5 : Une chaîne de blocage

Les chaînes de blocage ont toujours pour source une tâche émettrice de synchronisation non active « tardive » par rapport à la tâche réceptrice associée. Cette tâche, si elle est émettrice, peut

bloquer différentes tâches réceptrices qui lui sont associées, directement ou indirectement. Ce qu'il est important de constater, c'est qu'on interdit aux tâches d'attendre une synchronisation à l'intérieur d'une section critique, ce qui implique qu'aucune ressource n'est détenue par une tâche bloquée, qui ne pourra pas bloquer d'autre tâche de cette manière.

Nous pouvons formaliser les chaînes de blocage de cette manière :

L'ensemble des blocages potentiels directs dus à l'attente d'un message d'une tâche non (ré)activée au temps t est donné par un ensemble de triplets {tache bloquée, numéro relatif de l'attente, tâche bloquante}:

$$BD(t) = \left\{ \{i, k, j\} / \left(k + a_{i,j} \left\lfloor \frac{t - r_i}{T_i} \right\rfloor > e_{j,i} \left\lfloor \frac{T_j + t - r_j}{T_j} \right\rfloor \right) \wedge t \geq r_i \right\}$$

La valeur $a_{i,j} \left\lfloor \frac{t - r_i}{T_i} \right\rfloor + k$ correspond au numéro absolu de l'attente $A_{i,j,k}$ lors de l'activation en cours de τ_i au temps t, et $e_{j,i} \left\lfloor \frac{T_j + t - r_j}{T_j} \right\rfloor$ est le nombre maximal de messages émis par les occurrences successives de τ_j jusqu'au temps t, y compris par l'occurrence de l'activation courante. Si le nombre de messages attendus est supérieur au nombre de messages émissibles d'ici la prochaine activation de la tâche émettrice, il y a blocage potentiel de la tâche réceptrice au point $A_{i,j,k}$: cela signifie qu'en aucun cas la tâche τ_i ne pourra être avancée au temps t au–delà de $CA_{i,j,k}$.

L'ensemble BD(t) peut contenir plusieurs fois la tâche τ_i sur la première composante des triplets le composant puisqu'une tâche peut attendre des messages de plusieurs tâches non encore (ré)activées.

L'ensemble de toutes les tâches potentiellement bloquées directement ou indirectement par attente de message est produit par induction structurelle des tâches sur BD(t) de la manière suivante :

$$B(t) = BD(t) \cup \left\{ \{i, p, j\} / t \geq r_i \wedge \left(\exists \{j, q, k\} \in B(t) / CE_{j,i,M_{i,j}(p,t)} > CA_{j,k,q} \right) \right\}$$

Rappel : $M_{i,j}(p,t)$ est le numéro relatif au corps de la tâche τ_j de l'émission correspondant à la $p^{ième}$ réception par τ_i située dans l'activation en cours en temps t.

La définition constructive de B(t) s'écrit comme le point fixe de la suite convergente suivante :

$$B_0(t) = BD(t),\ B_m(t) = B_{m-1}(t) \cup \left\{ \{i, p, j\} / t \geq r_i \wedge \left(\exists \{j, q, k\} \in B_{m-1}(t) / CE_{j,i,M_{i,j}(p,t)} > CA_{j,k,q} \right) \right\}$$

Un blocage potentiel {i,p,j} est dans B(t) si il est dans BD(t) ou si τ_j est bloquée avant le point d'émission vers τ_i qui correspond à l'attente $A_{i,j,p}$ (figure 6). En aucun cas la tâche τ_i ne pourra être avancée au delà de $CA_{i,j,p}$ au temps t. De la même façon que pour BD(t), une tâche peut être présente plusieurs fois dans B(t). La construction de B(t) est illustrée sur la figure 6.

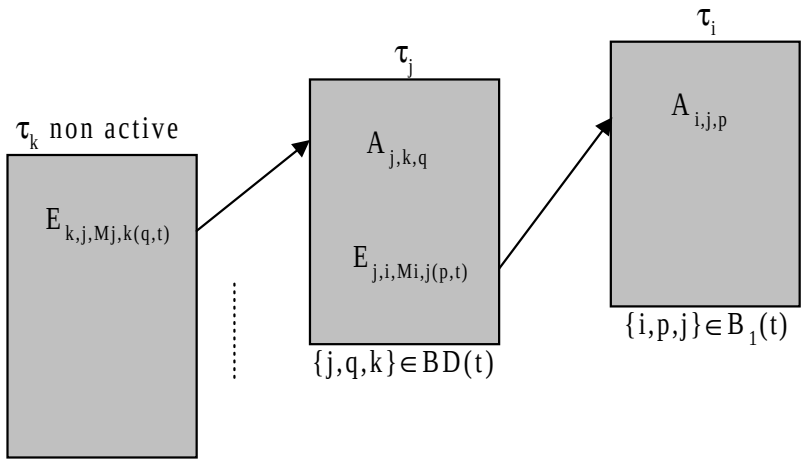

Figure 6 : Illustration de B(t)

Remarque : Les chaînes de blocage ne peuvent contenir à chaque instant t que des tâches présentes sur la première composante d'un triplet de B(t).

4.2 Périodicité des chaînes de blocage

Dans ce chapitre nous montrons des propriétés de cyclicité des chaînes de blocage.

Le lemme 1 montre que les tâches potentiellement bloquées à une certaine date le sont aussi une métapériode plus tard, et qu'elles le sont aux mêmes endroits.

Lemme 1 : $\forall \Delta \geq 0$, $B(\Delta) \subseteq B(P+\Delta)$

Preuve :

Dans une première partie nous montrons que $BD(\Delta) \subseteq BD(P+\Delta)$

Soit $\{i,k,j\} \in BD(\Delta)$, on a d'une part $\Delta \geq r_i$ donc $P+\Delta \geq r_i$

De plus $k + a_{i,j}\left\lfloor \frac{\Delta - r_i}{T_i} \right\rfloor > e_{j,i}\left\lfloor \frac{T_j + \Delta - r_j}{T_j} \right\rfloor$ **(1)**, d'après la CN 1, $\frac{e_{j,i}}{T_j} = \frac{a_{i,j}}{T_i}$ d'où $\frac{Pe_{j,i}}{T_j} = \frac{Pa_{i,j}}{T_i}$, ce qui nous permet d'écrire **(1)** $\Leftrightarrow a_{i,j}\frac{P}{T_i} + k + a_{i,j}\left\lfloor \frac{\Delta - r_i}{T_i} \right\rfloor > e_{j,i}\frac{P}{T_j} + e_{j,i}\left\lfloor \frac{T_j + \Delta - r_j}{T_j} \right\rfloor$ **(1)**, or $\frac{Pe_{j,i}}{T_j} = \frac{Pa_{i,j}}{T_i}$ sont des entiers.

D'où **(1)** $\Leftrightarrow k + a_{i,j}\left\lfloor \frac{P + \Delta - r_i}{T_i} \right\rfloor > e_{j,i}\left\lfloor \frac{P + T_j + \Delta - r_j}{T_j} \right\rfloor$.

De plus $P+\Delta \geq r_i$, ce qui signifie que $\{i,k,j\} \in BD(P+\Delta)$, d'où $BD(\Delta) \subseteq BD(P+\Delta)$.

Nous montrons que $B(\Delta) \subseteq B(P+\Delta)$ par induction :

$B_0(\Delta) \subseteq B_0(P+\Delta)$

Supposons $B_m(\Delta) \subseteq B_m(P+\Delta)$

Soit $\{i,p,j\} \in B_{m+1}(\Delta)$,

ou bien $\{i,p,j\} \in B_m(\Delta) \subseteq B_m(P+\Delta)$ (hypothèse de récurrence),

ou bien $\exists \{j,q,k\} \in B_m(\Delta)$ / $CE_{j,i,M_{i,j}(p,\Delta)} > CA_{j,k,q}$ et $\Delta \geq r_i$. On a donc d'une part $P+\Delta \geq r_i$, et d'autre part $\{j,q,k\} \in B_m(P+\Delta)$ (hypothèse de récurrence) et comme $M_{i,j}(p,\Delta) = M_{i,j}(p,P+\Delta)$

pour $\Delta \geq r_i$ car $M_{i,j}$ est périodique sur P, on a $CE_{j,i,M_{i,j}(p,P+\Delta)} > CA_{j,k,q}$, ce qui implique $\{i,p,j\} \in B_{m+1}(P+\Delta)$.

C.Q.F.D.

Le lemme 2 montre qu'à partir de r+P, les tâches potentiellement bloquées le sont aussi la métapériode précédente, de plus elles sont potentiellement bloquées aux mêmes endroits.

Lemme 2 : $\forall \Delta \geq 0$, $B(r+P+\Delta) \subseteq B(r+\Delta)$

Soit $\{i,k,j\} \in BD(r+P+\Delta)$, alors $k + a_{i,j}\left\lfloor \frac{r+P+\Delta-r_i}{T_i} \right\rfloor > e_{j,i}\left\lfloor \frac{r+P+\Delta+T_j-r_j}{T_j} \right\rfloor$ **(1)** or $P=PPCM(T_i)$ donc $\frac{P}{T_i}$ et $\frac{P}{T_j}$ sont des entiers. On a donc :

(1) $\Leftrightarrow$ $k + a_{i,j}\frac{P}{T_i} + a_{i,j}\left\lfloor \frac{r+\Delta-r_i}{T_i} \right\rfloor > e_{j,i}\frac{P}{T_j} + e_{j,i}\left\lfloor \frac{r+\Delta+T_j-r_j}{T_j} \right\rfloor$ or d'après la CN 1, $\frac{e_{j,i}}{T_j} = \frac{a_{i,j}}{T_i}$

donc **(1)** $\Leftrightarrow$ $k + a_{i,j}\left\lfloor \frac{r+\Delta-r_i}{T_i} \right\rfloor > e_{j,i}\left\lfloor \frac{r+\Delta+T_j-r_j}{T_j} \right\rfloor$.

De plus $r+\Delta \geq r_i$, d'où $BD(r+P+\Delta) \subseteq BD(r+\Delta)$.

Nous montrons que $B(r+P\Delta) \subseteq B(r+\Delta)$ par induction :

$B_0(r+P+\Delta) \subseteq B_0(r+\Delta)$

Supposons $B_m(r+P+\Delta) \subseteq B_m(r+\Delta)$

Soit $\{i,p,j\} \in B_{m+1}(r+P+\Delta)$,

ou bien $\{i,p,j\} \in B_m(r+P+\Delta) \subseteq B_m(r+\Delta)$,

ou bien $\exists \{j,q,k\} \in B_m(r+P+\Delta)$ / $CE_{j,i,M_{i,j}(p,r+P+\Delta)} > CA_{j,k,q}$ et $r+P+\Delta \geq r_i$. D'une part on a $r+\Delta \geq r_i$, d'autre part $\{j,q,k\} \in B_m(r+\Delta)$ et comme $M_{i,j}(p,r+\Delta) = M_{i,j}(p,r+P+\Delta)$ pour $r+\Delta \geq r_i$, $CE_{j,i,M_{i,j}(p,r+\Delta)} > CA_{j,k,q}$, ce qui implique $\{i,p,j\} \in B_{m+1}(r+\Delta)$.

C.Q.F.D.

Les deux lemmes précédents nous permettent de dire qu'à partir de la date r, il y a cyclicité des blocages potentiels sur la métapériode P.

Proposition 1 : $\forall \Delta \geq 0$, $B(r+P+\Delta) = B(r+\Delta)$

La preuve découle des lemmes 1 et 2.

4.3 Date d'occurrence du dernier temps creux acyclique

Nous rappelons que nous travaillons à charge 100%, donc que tous les temps creux sont acycliques. Nous montrons dans ce paragraphe que le dernier temps creux a lieu avant la date r+P. Nous introduisons un certain nombre de fonctions qui nous permettent de décrire l'état instantané du système.

4.3.1 Les fonctions de charge instantanée

$C_{start}(t,i) = \begin{cases} C_i \text{ si } t = r_i \\ 0 \text{ sinon} \end{cases}$ Charge instantanée initiale induite par τ_i.

Donc $\forall t > r_i$, $C_{start}(t,i) = 0$

Nous définissons $C_{start}(t)$ comme la somme des charges instantanées initiales au temps t :

$C_{start}(t) = \sum_{i=1}^{n} C_{start}(t,i)$ charge instantanée initiale au temps t

Donc $\forall t > r$, $C_{start}(t) = 0$

De la même façon que C_{start}, nous définissons $C_{réac}(t,i)$ qui est la charge instantanée de réactivation de τ_i au temps t :

$$C_{réac}(t,i) = \begin{cases} C_i \text{ si } ((t - r_i) \equiv T_i) = 0 \ \wedge \ t \neq r_i \\ 0 \text{ sinon} \end{cases}$$

$C_{réac}(t) = \sum_{i=1}^{n} C_{réac}(t,i)$ charge instantanée de réactivation au temps t

On peut d'ores et déjà remarquer que la fonction $C_{réac}(t)$ est périodique de période P pour t>r et que $C_{start}(t)$ est nulle pour t>r.

La charge instantanée de blocage $C_{bloqué}(t,i)$ induite par une tâche τ_i est la charge maximale de τ_i potentiellement bloquée au temps t.

$C_{bloqué}(t,i) = \max\{CA_{i,j,k} \ \forall \{i,j,k\} \in B(t)\}$

La charge bloquée est définie par $C_{bloqué}(t) = \sum_{i=1}^{n} C_{bloqué}(t,i)$

Lemme 3 : $\forall \Delta \geq 0$, $C_{bloqué}(P+\Delta) \geq C_{bloqué}(\Delta)$ et $C_{bloqué}(r+\Delta) = C_{bloqué}(r+P+\Delta)$

Preuve :

Première inéquation : découle du lemme 1

Deuxième inéquation : découle de la proposition 1

C.Q.F.D.

Enfin nous définissons récursivement la charge instantanée totale $C_{rest}(t)$ qui est le nombre d'unités de temps de traitement du système de tâches restant à effectuer au temps t lorsque le système est ordonnancé au plus tôt.

$C_{rest}(0) = C_{start}(0)$

$C_{rest}(t) = C_{start}(t) + C_{réac}(t) + C_{rest}(t-1) - \gamma(t-1)$

$$\text{avec } \gamma(t) = \begin{cases} 1 \text{ si } C_{rest}(t) > C_{bloqué}(t) \text{ (i.e si une tâche peut travailler à l'instant t)} \\ 0 \text{ si } C_{rest}(t) = C_{bloqué}(t) \end{cases}$$

$C_{rest}(t)$ définit la charge restant à traiter pour un système ordonnancé par n'importe quel ordonnancement au plus tôt. Le système est globalement bloqué à l'instant t si et seulement si la charge restant à traiter en t est égale à $C_{bloqué}(t)$. En effet dans ce cas les seules tâches non terminées sont toutes arrivées à un point d'attente.

4.3.2 Etude des temps creux

Le lemme 4 montre que le travail restant à effectuer en une date $t \geq r+P$ est supérieur ou égal au travail restant une métapériode plus tôt. Cela s'explique par le fait qu'en P unités de temps, on ne peut pas traiter plus de P unités de temps induites par les réactivations des tâches à cause de la charge de 100%.

Lemme 4 : $\forall \Delta \geq 0$, $C_{rest}(r+P+\Delta) \geq C_{rest}(r+\Delta)$

Preuve : $C_{rest}(t) = C_{start}(t) + C_{réac}(t) + C_{rest}(t-1) - \gamma(t-1)\ \forall t \geq 1$

$$\Rightarrow \sum_{i=1}^{P} C_{rest}(r+\Delta+i) = \sum_{i=1}^{P} C_{start}(r+\Delta+i) + C_{réac}(r+\Delta+i) + C_{rest}(r+\Delta+i-1) - \gamma(r+\Delta+i-1)$$

or $\forall i \in [1..P]$, $r+\Delta+i \geq r \Rightarrow C_{start}(r+\Delta+i) = 0$

$$\Rightarrow \sum_{i=1}^{P} C_{rest}(r+\Delta+i) = \sum_{i=1}^{P} C_{réac}(r+\Delta+i) + C_{rest}(r+\Delta+i-1) - \gamma(r+\Delta+i-1)$$

Toute tâche est activée $\frac{P}{T_i}$ fois en P unités de temps et va induire une charge de $P\frac{C_i}{T_i}$ unités de temps. Comme la charge est de 100%, les réactivations des tâches vont induire P unités de temps de charge en P unités de temps.

d'où $\sum_{i=1}^{P} C_{rest}(r+\Delta+i) = P + \sum_{i=1}^{P} C_{rest}(r+\Delta+i-1) - \gamma(r+\Delta+i-1)$

i.e. $\sum_{i=1}^{P} C_{rest}(r+\Delta+i) = P + \sum_{i=0}^{P-1} C_{rest}(r+\Delta+i) - \gamma(r+\Delta+i)$ **(1)**

or $0 \leq \gamma(t) \leq 1\ \forall t \geq 0$

$$\Rightarrow \sum_{i=1}^{P} C_{rest}(r+\Delta+i) \geq P - P + \sum_{i=0}^{P-1} C_{rest}(r+\Delta+i)$$

$\Rightarrow C_{rest}(r+P+\Delta) \geq C_{rest}(r+\Delta)$

C.Q.F.D.

Le lemme 5 montre que si il existe un temps creux au temps $r+\Delta$, il n'y en a pas au temps $r+P+\Delta$.

Lemme 5 : $\forall \Delta \geq 0$, $\gamma(r+\Delta)=0 \Rightarrow \gamma(r+\Delta+P)=1$

Preuve :

$$\sum_{i=1}^{P} C_{rest}(r+\Delta+i) = P + \sum_{i=0}^{P-1} C_{rest}(r+\Delta+i) - \gamma(r+\Delta+i) \qquad \textbf{(1)}$$

or par hypothèse $\gamma(r+\Delta)=0$

$$\Rightarrow \sum_{i=1}^{P} C_{rest}(r+\Delta+i) = P + C_{rest}(r+\Delta) + \sum_{i=1}^{P-1} C_{rest}(r+\Delta+i) - \gamma(r+\Delta+i)$$

or $0 \leq \gamma(t) \leq 1\ \forall t \geq 0$

$$\Rightarrow \sum_{i=1}^{P} C_{rest}(r+\Delta+i) \geq 1 + C_{rest}(r+\Delta) + \sum_{i=1}^{P-1} C_{rest}(r+\Delta+i)$$

$\Rightarrow C_{rest}(r+P+\Delta) \geq 1+C_{rest}(r+\Delta)$ or $\gamma(r+\Delta)=0 \Leftrightarrow C_{rest}(r+\Delta)=C_{bloqué}(r+\Delta)$

$\Rightarrow C_{rest}(r+P+\Delta) \geq 1+C_{bloqué}(r+\Delta)$ or d'après le lemme 3 $C_{bloqué}(r+P+\Delta)=C_{bloqué}(r+\Delta)$ d'où $C_{rest}(r+P+\Delta) \geq 1+C_{bloqué}(r+P+\Delta)$ et par définition $\gamma(r+\Delta+P)=1$.

C.Q.F.D.

Le lemme 6 montre que si il n'y a pas de temps creux en une date située après r, il n'y en a pas non plus une métapériode plus tard.

Lemme 6 : $\forall \Delta \geq 0, \gamma(r+\Delta)=1 \Rightarrow \gamma(r+P+\Delta)=1$

Preuve :

$\gamma(r+\Delta)=1 \Rightarrow C_{rest}(r+\Delta) > C_{bloqué}(r+\Delta)$

De plus $C_{rest}(r+P+\Delta) \geq C_{rest}(r+\Delta)$ (lemme 4) et $C_{bloqué}(r+\Delta)=C_{bloqué}(r+P+\Delta)$ (lemme 3)

$\Rightarrow C_{rest}(r+P+\Delta) > C_{bloqué}(r+P+\Delta)$

C.Q.F.D.

Proposition 2 : Dans les ordonnancements au plus tôt de charge 100%, il ne peut y avoir de temps creux à partir de la date r+P.

Preuve :

Comme par définition $\gamma(t)$ vaut soit 0, soit 1, les lemmes 5 et 6 montrent que $\gamma(r+\Delta+P)=1$ $\forall \Delta \geq 0$.

C.Q.F.D.

5. Cyclicité des ordonnancements au plus tôt

Soit t_c la date du dernier temps creux : $\forall t > t_c$, il existe τ_i telle que $C_{rest}(t,i) > C_{bloqué}(t,i)$. D'après la proposition 2, $t_c < r + P$ pour tout ordonnancement valide au plus tôt. Donc $\forall t > t_c$, $C_{rest}(t) = C_{start}(t) + C_{réac}(t) + C_{rest}(t-1) - 1$. De plus il se peut qu'il n'y ait aucun temps creux dans l'ordonnancement, dans ce cas, nous considérons que le dernier temps creux a eu lieu à l'unité de temps précédent 0 ($t_c=-1$).

Nous étendons donc la définition de $C_{rest}(t)$ sur l'ensemble des entiers relatifs en posant $C_{rest}(t)=0$ $\forall t<0$.

Puisque au temps t_c il y avait un temps creux, $C_{rest}(t_c)=C_{bloqué}(t_c)$ donc $C_{rest}(t_c+1)=C_{start}(t_c+1)+C_{réac}(t_c+1)+ C_{bloqué}(t_c)$.

Lemme 7 : $\forall i \in \{1..n\}, r_i < t_c+T_i$

Preuve :

Supposons qu'$\exists j / r_j \geq t_c+T_j$.

$C_{rest}(t) = C_{start}(t) + C_{réac}(t) + C_{rest}(t-1) - \gamma(t-1)$ $\forall t \geq 1$

$$\Rightarrow \sum_{i=1}^{P} C_{rest}(t_c+i) = \sum_{i=1}^{P} C_{start}(t_c+i) + C_{réac}(t_c+i) + C_{rest}(t_c+i-1) - \gamma(t_c+i-1)$$

Toute tâche est activée $\frac{P}{T_i}$ fois en P unités de temps et va induire une charge de $P\frac{C_i}{T_i}$ unités de temps. Comme la charge est de 100%, mais que τ_j n'est pas activée pendant au moins sa première période, les activations et réactivations des tâches vont induire au plus $P-C_i$ unités de temps de charge en P unités de temps.

$$\text{d'où } \sum_{i=1}^{P} C_{rest}(t_c+i) \leq P - C_i + \sum_{i=1}^{P} C_{rest}(t_c+i-1) - \gamma(t_c+i-1)$$

$$\text{et } \sum_{i=1}^{P} C_{rest}(t_c+i) \leq P - C_i + \sum_{i=0}^{P-1} C_{rest}(t_c+i) - \gamma(t_c+i)$$

Par définition $\gamma(t_c)=0$ et $\forall i>0$ $\gamma(t_c+i)=1$

$\Rightarrow \sum_{i=1}^{P} C_{rest}(t_c+i) \le P - P + 1 - C_i + \sum_{i=0}^{P-1} C_{rest}(t_c+i)$

$\Rightarrow C_{rest}(t_c+P) \le 1-C_i+C_{rest}(t_c)$

$\Rightarrow C_{rest}(t_c+P) \le 1-C_i+C_{bloqué}(t_c)$ or d'après le lemme $C_{bloqué}(t_c) \le C_{bloqué}(t_c+P)$ d'où

$C_{rest}(t_c+P) \le 1-C_i+C_{bloqué}(t_c+P)$

Si $C_i>1$, alors $C_{rest}(t_c+P)<C_{bloqué}(t_c+P)$ d'où contradiction.

Si $C_i=1$, alors $C_{rest}(t_c+P) \le C_{bloqué}(t_c+P)$ donc $C_{rest}(t_c+P)=C_{bloqué}(t_c+P)$, ce qui signifie qu'il y a un temps creux en t_c+P d'où contradiction.

C.Q.F.D.

Lemme 8 : $\forall i \in \{1..N\}$, $C_{start}(t_c+1,i)+C_{réac}(t_c+1,i) = C_{réac}(t_c+P+1,i)$

Preuve : Découle du lemme 7.

Proposition 3 : Tout ordonnancement au plus tôt valide jusqu'à t_c+P laisse le système de tâches dans le même état au temps t_c+P+1 qu'au temps t_c+1.

Preuve :

$C_{rest}(t_c+1)=C_{bloqué}(t_c)+C_{start}(t_c+1)+C_{réac}(t_c+1)$ car par définition $\gamma(t_c)=0$

$C_{rest}(t_c+P+1)=C_{rest}(t_c+P)+C_{start}(t_c+P+1)+ C_{réac}(t_c+P+1)-1$ car par définition $\gamma(t_c+P)=1$

D'après le lemme 7, $r_i<t_c+T_i \ \forall i \in 1..n$, donc $C_{start}(t_c+P+1)=0$

D'après le lemme 8 $C_{réac}(t_c+P+1)= C_{start}(t_c+1)+C_{réac}(t_c+1)$, de plus cette charge est induite par les mêmes réveils de tâches car P est la métapériode du système de tâches.

Par suite $C_{rest}(t_c+P+1)=C_{rest}(t_c+P)+C_{start}(t_c+1)+C_{réac}(t_c+1)-1$

Comme le processeur a travaillé pendant exactement $P-1$ unités de temps entre t_c et $P+t_c$ (un seul temps creux : celui situé au temps t_c), on a $C_{rest}(t_c+P)=P-P+1+C_{rest}(t_c)=C_{rest}(t_c)+1=C_{bloqué}(t_c)+1$. Or d'après le lemme 3, $C_{bloqué}(t_c+P) \ge C_{bloqué}(t_c)$. D'où $C_{rest}(t_c+P)=C_{bloqué}(t_c)+1 \le C_{bloqué}(t_c+P)+1$.

Si $C_{rest}(t_c+P)<C_{bloqué}(t_c+P)+1$, alors il y a un temps creux en t_c+P, ce qui est impossible. Donc $C_{rest}(t_c+P)=C_{bloqué}(t_c)+1=C_{bloqué}(t_c+P)+1$ **(1)**.

Donc $C_{rest}(t_c+P+1)=C_{bloqué}(t_c)+C_{start}(t_c+1)+C_{réac}(t_c+1)$ avec les réactivations qui sont les mêmes que les activations /réactivations en t_c.

L'équation **(1)** implique que $C_{bloqué}(t_c)=C_{bloqué}(t_c+P)$, ce qui signifie, comme $B(t_c) \subseteq B(t_c+P)$, qu'il y a les mêmes blocages en t_c qu'en t_c+P.

C.Q.F.D.

Théorème : Un ordonnancement au plus tôt d'un système de tâches de charge 1 est valide si et seulement si il est valide sur $[0..t_c+P+1]$ avec t_c date du dernier temps creux. De plus $t_c<r+P$.

Preuve :

$t_c<r+P$ découle de la proposition 2.

Sens du si : si l'ordonnancement n'est pas valide sur $[0..t_c+P+1]$ il est évident qu'il n'est pas valide (stabilité par préfixe des ordonnancements valides).

Sens du seulement si : découle de la proposition 3.

C.Q.F.D.

6. Conclusion

Nous avons montré que les dates d'occurrence de temps creux dans un ordonnancement au plus tôt pour un système de tâches de charge 100% sont bornées dans le temps par r+P. De plus nous avons montré que tout ordonnancement valide au plus tôt d'un tel système était cyclique de période P à partir de la date suivant l'occurrence du dernier temps creux.

Pour ordonnancer un système avec une politique d'ordonnancement au plus tôt (*Rate Monotonic First*, *Deadline Monotonic First*, *Earliest Deadline First*, ...) il suffit de l'ordonnancer jusqu'à r+P, à cette date, la date t_c du dernier temps creux est connue : on compte les temps creux, s'il n'y en a pas, on considère que $t_c=-1$, sinon c'est la date du dernier temps creux trouvé qui est t_c. Il ne reste plus qu'à ordonnancer, si ce n'est déjà fait (cas où $t_c<r$), jusqu'à t_c+P+1. La séquence MS^* est donnée par M = séquence sur $[0..t_c]$ et S séquence sur $[t_c+1..t_c+P+1]$. On peut remarquer qu'il existe des systèmes de tâches non synchrones au démarrage pour lesquels la séquence d'ordonnancement est directement S^*, avec S de longueur $PPCM(T_i)$.

Ce résultat est valable pour n'importe quel algorithme au plus tôt déterministe (i.e. face à la même configuration de tâches <compteurs ordinaux, horloges locales>, l'algorithme prend toujours la même décision), ce qui est le cas pour la plupart des algorithmes connus. De plus comme seules les charges et dates d'activations sont prises en compte pour la démonstration, elle s'applique sur les systèmes de tâches partageant des ressources, quel que soit le protocole de gestion de ressources utilisé (protocole à priorité plafond [2][5], protocole à priorité héritée [9], gestion des ressources par pile [1]). Ceci s'explique par le fait que mis à part le cas de l'interblocage, qui est repéré avant r+P, le blocage d'une tâche par l'attente d'une ressource ne peut pas globalement bloquer le système de tâches.

Ce résultat est utilisable non seulement lors de la validation hors–ligne d'algorithmes en–ligne mais aussi pour la recherche de séquences hors–ligne de séquences d'ordonnancement [4][6].

7. Bibliographie

[1] T.P. Baker, « Stack-Based Scheduling of Realtime Processes », Journal of Real-Time Systems, 3, 1991.

[2] M. Chen, K. Lin, « Dynamic Priority Ceilings : A Control Protocol for Real-Time Systems », Journal of Real-Time Systems, 2, 1990.

[3] H. Chetto, M. Silly, T. Bouchentouf, « Dynamic Scheduling of Real-Time Tasks Under Precedence Constraints », Journal of Real-Time Systems, 2, 1990.

[4] A. Choquet–Geniet, D. Geniet, F. Cottet, « Exhaustive Computation of the Scheduled Task Execution Sequences of a Real–Time Application », proc. 4th International Symposium on Formal Techniques in Real Time and Fault Tolerant Systems, Uppsala, Sweden, 09/11–13/96.

[5] J.B. Goodenough, L. Sha, « The Priority Ceiling Protocol : A Method for Minimizing the Blocking of High Priority Ada Tasks », Proc. 2nd ACM Int. Workshop Real–Time Ada Issues, 1988.

[6] E. Grolleau, A. Choquet-Geniet, F. Cottet, « Ordonnancement Optimal de Systèmes de Tâches Temps Réel à l'Aide de Réseaux de Petri », proc. AGIS'1997, 12/9-11/1997.

[7] J.Y.T. Leung, M.L. Merrill, « A Note on Preemptive Scheduling of Periodic Real–Time Tasks », Information Processing Letters 11 (3), pp 115–118, 1980.

[8] C.L. Liu, J.W. Layland, « Scheduling Algorithms for Multiprogramming in a Hard Real–Time Environment », journal of the ACM, 20 (1), pp 46–61, 1973.

[9] L. Sha, R. Rajkumar, J.P. Lehoczky, « Priority Inheritance Protocols : An Approach to Real–Time Synchronisation », IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175–1185, September 1990.

[10] J.A. Stankovic, « Misconception about Real–Time Computing », IEEE Computer Magazine, 21 (10), 1pp 0–19, 1988.

[11] J.A. Stankovic, M. Spuri, M. Di Natale, G. Buttazo, « Implications of Classical Results for Real-Time Systems », IEEE Computer, vol. 28, n. 6, pp1-25, 1995.