MARTIN Hervé
BOTHOREL Sébastien

Maîtrise Informatique
Année 2003 -2004

Travail d'Etude et de Recherche

Tests de faisabilité basés sur le taux d'occupation CPU des principaux algorithmes d'ordonnancement temps réel

état de l'art, propositions et implémentations dans le logiciel Cheddar

Maître de TER : Franck SINGHOFF
Université de Bretagne Occidentale

Sommaire

1) Introduction

Deux objectifs principaux étaient visés dans le TER que nous avons effectué. Le premier consistait en une recherche de tests de faisabilité basés sur le taux d'occupation CPU pour les politiques d'ordonnancements les plus connus dans un contexte monoprocesseur. Le deuxième était d'implanter ces tests dans le logiciel Cheddar. Ce dernier permet de modéliser et de simuler un jeu de tâches sous différentes politiques.

Rappelons ce qu'est un test de faisabilité (ou test d'ordonnançabilité). Il s'agit d'un test qui indique par un calcul simple (si possible), si les échéances des tâches peuvent être respectées par une politique et ce sans avoir besoin d'exécuter l'algorithme.

Les algorithmes d'ordonnancement visés par notre recherche étaient les suivants : Rate Monotonic, Deadline Monotonic, Earliest Deadline First, Least Laxity First, Fixed Preemptive Protocol, Non Preemptive Fixed Protocol et Round Robin. Pour chacun d'eux, un test devait être trouvé dans le cas préemptif et dans le cas non préemptif. Un algorithme préemptif signifie qu'une tâche élue peut perdre le processeur au profit d'une autre jugée plus prioritaire. Par contre, dans un algorithme non préemptif, l'ordonnanceur n'arrête pas une tâche élue.

Ce rapport va donc présenter les différents tests que nous avons pu collecter. Pour chaque politique d'ordonnancement, une définition est donnée ainsi que les tests de faisabilité dans les cas préemptif et non préemptif.

Un exemple d'utilisation accompagne certains tests. De plus, une réflexion sur la validité et l'intérêt de certains tests est donnée. Ainsi, une explication accompagne des tests et parfois nous montrons les intérêts ou au contraire les insuffisances que nous avons pu remarquer.

Comme nous avons implanté ces tests dans le logiciel Cheddar, nous en fournissons le code en Ada. Des photos d'écrans montrent des exemples d'utilisation des politiques avec le logiciel et les résultats des tests correspondants.

Pour certaines politiques visées, aucun test n'a pu être trouvé. Toutefois, pour une partie d'entre elles nous émettons des tests possibles en se basant sur les tests existants. Pour une autre partie des politiques non trouvées, nous n'avons pas pu établir de test. Dans ce cas nous expliquons les difficultés rencontrées.

Ce rapport est divisé en trois parties.

La première partie correspond aux recherches que nous avons mené et donnent les différents tests que nous avons pu collecter.

La seconde partie présente nos propositions sur les tests manquants ou, quand aucun test n'a pu être établi, exprime les difficultés que nous avons rencontré à déterminer ces tests.

La troisième partie concerne l'implémentation de certaines politiques dans le logiciel CHEDDAR.

2) Les tests de faisabilité

Cette partie relate la recherche que nous avons mené sur les tests de faisabilité basés sur le taux d'occupation processeur. Nous passons en revue les différents tests rapportés. Pour chaque politique nous rappelons la manière dont une priorité est fixée. Des exemples d'utilisation des tests accompagnent les formules. De plus, nous ajoutons parfois des réflexions personnelles sur ces tests. A la fin de cette partie du rapport, un tableau récapitule l'ensemble des tests trouvés.

2.1) Rate Monotonic

C'est un algorithme qui est basé sur des priorités statiques et qui est optimal. Une tâche dispose d'une priorité fixe qui est inversement proportionnelle à la période. Nous présentons les tests de faisabilité dans le cas préemptif et non préemptif.

2.1.1) RM préemptif

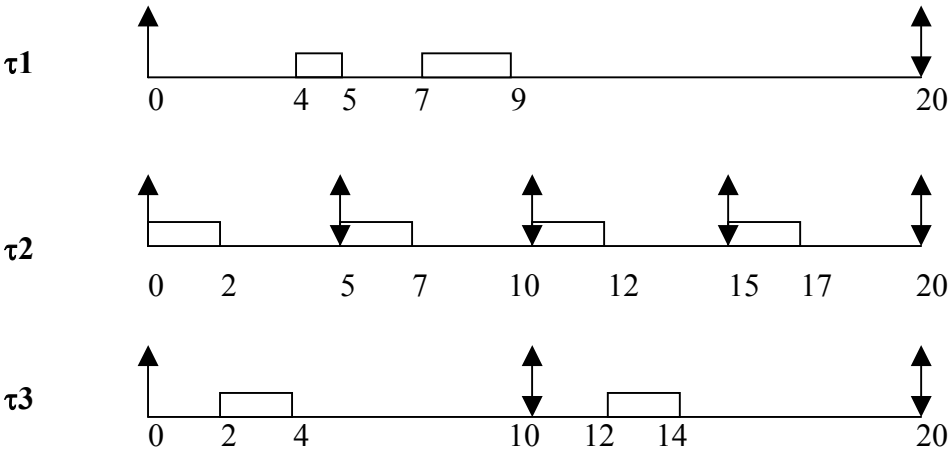
Une **condition suffisante** a été définie par **[LL 73].** L'ordonnancement Rate Monotonic d'un ensemble de n tâches périodiques est faisable si le facteur d'utilisation U vérifie la relation suivante :

$$U = \sum_{i=1}^{n} Ci / Pi$$

$$U \leq n * (2^{1/n} - 1) \qquad (1)$$

L'exemple suivant présente un ordonnancement Rate Monotonic pour trois tâches périodiques.

	Ci	Pi
τ1	3	20
τ2	2	10
τ3	2	5



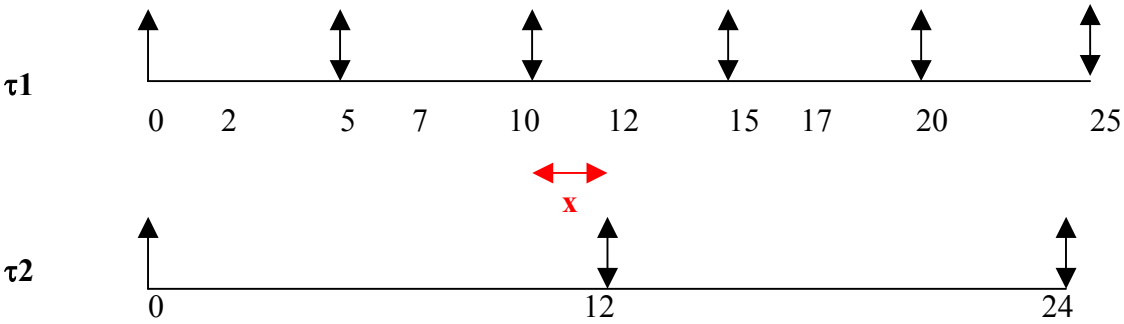
Les trois tâches respectent leur échéance temporelle. La condition de faisabilité est vérifiée.

$3/20 + 2/10 + 2/5 \leq 3 * (2^{1/3} - 1)$

0,75 ≤ 0,77 (test respecté)

Nous nous sommes intéressés à la manière dont Liu et Layland **[LL73]** ont défini la borne du test de faisabilité et plus particulièrement pourquoi le taux d'utilisation ne peut pas atteindre 100 % avec plus d'une tâche. Nous présentons leur réflexion au travers d'un exemple de deux tâches périodiques **τ1** et **τ2** avec **τ1** prioritaire par rapport à **τ2.**

	Ci	**Pi**
τ1	C1	5
τ2	C2	12



L'algorithme Rate Monotonic est à priorité fixe donc il ne s'occupe pas de savoir si une tâche se rapproche de son échéance. Dans l'exemple, la tâche **τ1** a une priorité plus forte que la tâche **τ2**. Au temps 10, même si la tâche **τ1** a une échéance plus lointaine que **τ2**, elle aura toujours la priorité. Ceci implique que la capacité de **τ2** (C2) est contrainte par la capacité de **τ1**. Liu et Layland essayent de maximiser la capacité de la tâche **τ2** en fonction d'une valeur affectée à la capacité de **τ1**. Deux cas peuvent se présenter pour la valeur de C1:

- soit C1 est suffisamment petit pour accepter que C2 puisse se placer pendant mais également à la fin de sa période. Alors C2 peut disposer d'une valeur importante.
- soit C1 est trop grand pour que C2 puisse se placer à la fin de sa période, ce qui limite la valeur de C2.

Dans l'exemple la capacité de **τ1** est soit inférieure/égale à x, soit supérieure/égale à x. La valeur de x se détermine de la manière suivante :

$x = P2 - P1 * \lfloor P2/P1 \rfloor = 12 - (5 * \lfloor 12/5 \rfloor) = 12 - (5*2) = 2$

Donc x=2 dans l'exemple.

• Soit la capacité de **τ1** est inférieure ou égale à x et dans ce cas la plus grande valeur de C2 vaut :

$$C2 = P2 - C1 * \lceil P2/P1 \rceil$$
$$= P2 - C1 * \lceil 12/5 \rceil$$
$$= \mathbf{12 - (C1*3)}$$

On peut ainsi calculer les différentes valeurs du taux d'utilisation en fonction des valeurs prises par C1 (0 à 2).

- Pour C1=0 ⇒ C2 = 12-(0*3) = 12
  U = 12/12 + 0/5 = 12/12 = 1 (100% mais la tâche **τ1** n'existe plus)
- Pour C1=1 ⇒ C2 = 12-(1*3) = 9
  U = 9/12 + 1/5= (45+12)/60 = 57/60 = 0,95 (<100%)
- Pour C1=2 ⇒ C2 = 12-(2*3) = 6
  U= 6/12 * 2/5 = (30+24)/60= 54 /60 =0,90 (<100%)

Liu et Layland transforment la formule du taux d'utilisation pour deux tâches (C1/P1 + C2/P2) en remplaçant la valeur de C2 par la formule de la plus grande valeur de C2.

• Soit la capacité de **τ1** est supérieure ou égale à x et dans ce cas la plus grande valeur de C2 vaut :

$$\begin{aligned} C2 &= P1 * \lfloor P2/P1 \rfloor - C1 * \lfloor P2/P1 \rfloor \\ &= 5 * \lfloor 12/5 \rfloor - C1 * \lfloor 12/5 \rfloor \\ &= 5*2 - C1*2 \\ &= \mathbf{10-C1*2} \end{aligned}$$

Si la valeur de C1 est supérieure à celle de C2, la tâche **τ2** ne peut pas se situer dans la partie entre 10 et 12 car forcément elle est prise par **τ1**. Ainsi C2 ne pourra se placer que entre 0 et 10 au maximum (enlevé de C1 multiplié par le nombre de fois qu'elle est présente).

Les valeurs de C1, dans ce cas, sont comprises entre 2 et 5 et le taux d'utilisation U vaut alors :

- Pour C1=2 ⇒ C2= 10-(2*2) = 6
  U = 6/12 + 4/5 = (30+24)/60 = 54/60 = 0,90 (<100%)
- Pour C1=3 ⇒ C2=10-(3*2) = 4
  U = 4/12 + 3/5 = (20+36)/60 = 56/60 =0,93 (<100%)
- Pour C1=4 ⇒ C2=10-(4*2) = 2
  U = 2/12 + 4/5 = (10+48)/60= 58 /60 =0,96 (<100%)
- Pour C1=5 ⇒ Cb=10-(5*2)=0
  U = 0/12 + 5/5 = 5/5 = 1 (100% mais la tâche b n'existe plus)

Ici encore, Liu et Layland transforment la formule du taux d'utilisation de deux tâches par la formule de la plus grande valeur de C2.

Liu et Layland effectuent une égalité entre les deux nouvelles formules des taux d'utilisation afin d'obtenir une moyenne du taux d'utilisation et arrivent à la valeur suivante pour deux tâches: $\mathbf{2 * (2^{1/2} - 1)}$. Ensuite, ils étendent leur réflexion à n tâches et obtiennent la borne :

$$\mathbf{n * (2^{1/n} - 1).}$$

Il est alors intéressant de comparer les taux d'utilisation obtenus dans l'exemple avec la borne $n * (2^{1/n} - 1)$. Pour n=2 la, borne arrive à la valeur 0,83. Ainsi, on constate facilement qu'il s'agit d'un test pessimiste, car les taux d'utilisations trouvés dans l'exemple dépassent la borne.

En plus du test de **[LL73]**, il existe également une **condition nécessaire et suffisante** basée sur le temps de réponse donnée dans **[JP86].** Sachant que le temps de réponse d'une tâche **τi** est borné supérieurement par :

$$RTi = \sum_{j=1}^{i} \lceil RTi/Pj \rceil * Pj + Ci \qquad (2)$$

l'ordonnancement de l'ensemble des tâches est faisable si et seulement si :

$$\forall i \ \ 1 \le i \le n \quad RTi \le Pi$$

2.1.2) RM non préemptif

Pour RM non préemptif, un travail a été effectué sur la recherche d'une condition d'ordonnançabilité **[CAR96]**. Celle-ci est basée sur l'étude du partage de ressources des algorithmes préemptifs, et en particulier des techniques à priorité héritée **[SRL 90].** Ainsi, les deux tests suivants présentent deux manières de vérifier l'ordonnancement :

1ère test :

$$\forall i \ \ 1 \le i \le n \quad \sum_{j=1}^{i} Cj/Pj + Bi/Pi \le i * (2^{1/i} - 1) \qquad (3)$$

2ème test:

$$\sum_{i=1}^{n} Ci/Pi + \max_{1 < i \le n} \{ Bi/Pi \} \le n * (2^{1/n} - 1) \qquad (4)$$

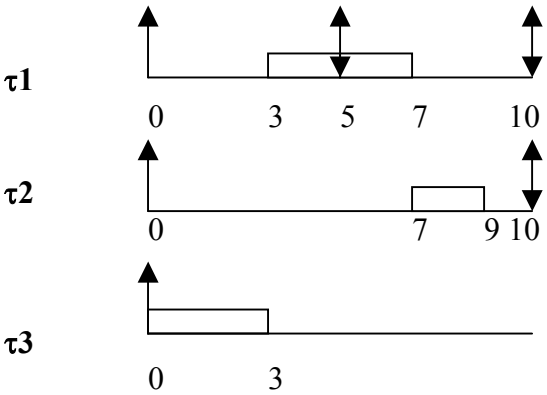
Avec

$$Bi = \max_{i+1 \le j \le n} \{ Cj \}$$

Bi représente la durée maximale du blocage d'un travail de la tâche **τi** par un travail d'une tâche de priorité inférieure, **τi+1**, …., **τn**. Le 1er test nécessite d'effectuer un calcul sur chaque tâche tandis que le 2ème test est un calcul global . Dans les deux cas, les tâches sont ordonnées selon leur priorité, c'est-à-dire de la plus petite à la plus grande période. Notons que **ces deux tests sont pessimistes** comme le montre l'exemple suivant.

	Ci	**Pi**
τ1	2	5
τ2	2	10
τ3	3	20

Dans cet exemple le premier test est uniquement effectué sur la tâche **τ2.** Dans ce cas, la période d'évaluation est 10**.** Les tâches **τ1** et **τ2** sont bloquées par une tâche de priorité inférieure.



Le premier test sur la tâche **τ2** n'est pas satisfait bien que la tâche **τ2** ne dépasse pas son échéance sur le schéma.

$2/5 + 2/10 + 3/10 \le 2 * (2^{1/2} - 1)$

$(4+2+3)/10 \le 0{,}83$

0,90 ≤ 0,83 (test non respecté)

Le deuxième test est global. Comme le test précédent, il n'est pas satisfait

$2/5 + 2/10 + 3/20 + \max(3/5+ 3/10) \leq 3 * (2^{1/3} - 1)$

$(8+4+3)/20 + 3/10 \leq 0,77$

$(14+6)/20 \leq 0,77$

$1 \leq 0,77$ (test non respecté)

Ces deux tests partent du principe qu'une tâche peut bloquer pendant toute sa durée, une tâche de priorité supérieure. En réalité, la durée de blocage n'est jamais aussi longue. Une tâche peut être bloquée par une tâche de priorité inférieure qu'un temps après que cette dernière ait démarré. Car si les deux tâches sont élues simultanément, ce sera toujours la plus prioritaire qui sera élue et donc il n'y a pas d'inversion de priorité.

Nous verrons plus loin dans le rapport que le test d'EDF non préemptif tient mieux compte de ce pire temps de déphasage.

2.2) Deadline Monotonic Préemptif

Nos recherches sur les tests de faisabilité dans le cas de l'algorithme Deadline Monotonic nous ont seulement permis de trouver des tests dans le cas préemptif. Après avoir rappelé la manière dont se calcule la priorité pour cet algorithme, nous présentons ces tests

L'algorithme Deadline Monotonic est un algorithme à priorité statique. La priorité d'une tâche est inversement proportionnelle à son échéance. Cet algorithme équivaut à Rate Monotonic quand l'échéance est égale à la période (échéance sur requête).

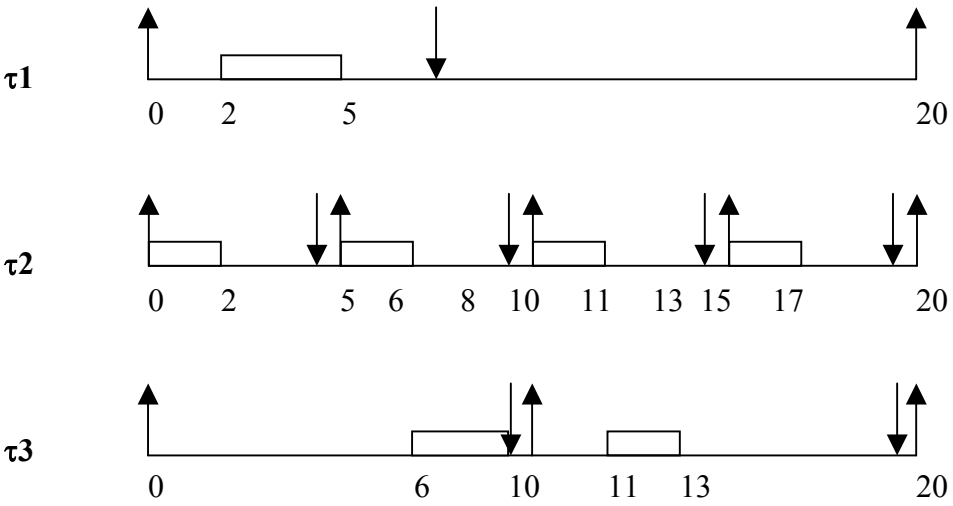
Dans le cas préemptif, **deux conditions suffisantes** existent.

La première condition suffisante a été donné dans **[LL73]**.

$$\sum_{i=1}^{n} Ci / Di \leq n * (2^{1/n} - 1) \qquad (5)$$

L'exemple suivant donne un ordonnancement Deadline Monotonic pour trois tâches périodiques :

	Ci	Di	Pi
τ1	3	7	20
τ2	2	4	5
τ3	2	9	10



Ce test ne valide pas l'ordonnançabilité du jeu de tâches bien qu'aucune tâche ne dépasse son échéance.

$3/7 + 2/4 + 2/9 \leq 3 * (2^{1/3} - 1)$

$(108+126+56)/252 \leq 0.77$

$290/252 \leq 0,77$

$1,15 \leq 0,77$ (non ordonnançable)

La seconde condition suffisante a été présentée dans **[AB90]** où les tâches sont ordonnées selon leur priorité :

$$\forall i\ 1 \le i \le n \quad Ci + \sum_{j=1}^{i-1} \lceil Di / Pj \rceil * Cj \le Di \qquad (6)$$

Ce test mesure l'interférence des processus de plus haute priorité dans l'exécution d'une tâche **τi** :

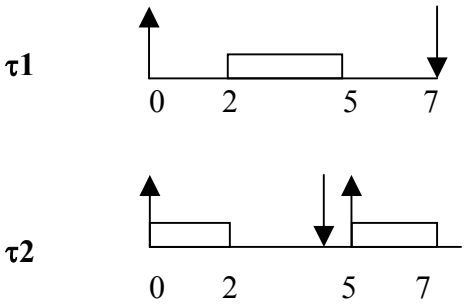
$$\sum_{j=1}^{i-1} \lceil Di/Pj \rceil * Cj$$

Il énonce que le temps de travail (Ci) de la tâche **τi** ajoutée à l'interférence imposée par les autres tâches ne doit jamais dépasser son échéance. Cette condition est suffisante mais pas nécessaire car la mesure de l'interférence est pessimiste : elle ne prend pas en compte les instants de déclenchement des différentes tâches.

Reprenons l'exemple précédent et appliquons le test à la tâche **τ1**. Dans le tableau suivant, les tâches sont rangées par ordre de priorité :

	priorité	**Ci**	**Di**	**Pi**
τ2	**1**	2	4	5
τ1	**2**	3	7	20
τ3	**3**	2	9	10

Le schéma suivant présente graphiquement le test pour la tâche **τ1**. La tâche **τ2** est plus prioritaire que la tâche **τ1** donc elle doit être prise en compte dans le calcul. Par contre la tâche **τ3** moins prioritaire n'interfère pas sur l'exécution de la tâche **τ1**. Le calcul du test est illustré par le schéma suivant :



Le test de faisabilité sur la tâche **τ2** ne provoque pas de débordement :

$C1 + \lceil D1/ P2 \rceil * C2 \le D1$
$3 + \lceil 7/ 5 \rceil * 2 \le 7$
$3 + 2 * 2 \le 7$
$7 \le 7$ (test satisfait)

2.3) Earliest Deadline First

L'algorithme Earliest Deadline First est basé sur des priorités dynamiques. La priorité maximale est accordée à la tâche dont l'échéance est la plus proche.

Nous avons trouvé des tests de faisabilité pour le cas préemptif et le cas non préemptif que nous présentons ci-après. Dans les deux cas, il faut distinguer les tâches à échéance sur requête et les tâches quelconques.

2.3.1) EDF préemptif

Pour des tâches à échéance sur requête une **condition nécessaire et suffisante** a été donnée par **[LL73]**. Celle-ci donne une borne sur le taux d'utilisation :

$$U = \sum_{i=1}^{n} Ci / Pi \qquad U \leq 1 \qquad (7)$$

A la différence de l'algorithme Rate Monotonic, on constate que pour l'algorithme EDF à échéance sur requête le taux d'utilisation peut atteindre 100%. En effet, il s'agit d'un algorithme à priorité dynamique, et une tâche voit sa priorité augmenter lorsqu'elle se rapproche de son échéance. Ainsi, l'utilisation du processeur est améliorée ; il y a moins de « temps morts ».

Pour des tâches quelconques **[LM 80]** a montré qu'une **condition suffisante** est :

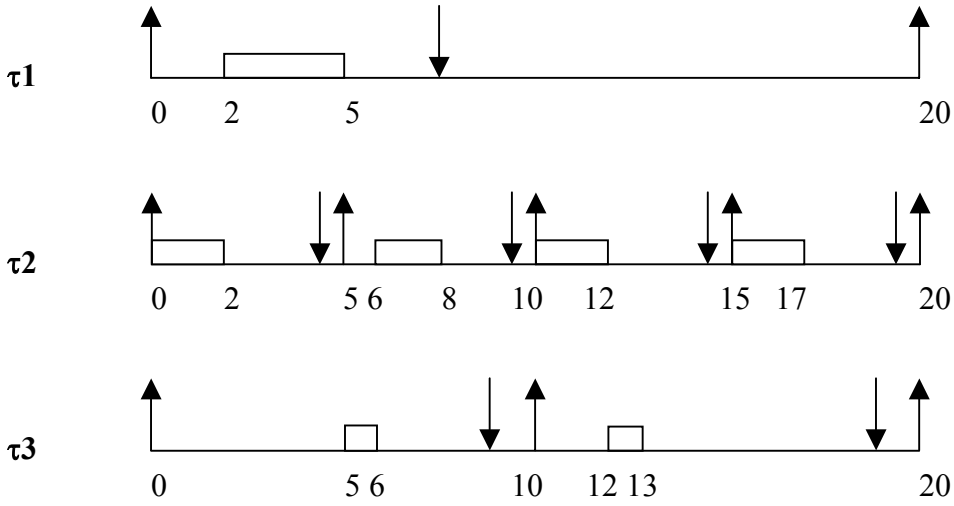
$$\sum_{i=1}^{n} Ci / Di \leq 1 \qquad (8)$$

et une **condition nécessaire** est celle donnée par la première formule :

$$U = \sum_{i=1}^{n} Ci / Pi \qquad U \leq 1$$

L'exemple suivant présente un ordonnancement de tâches quelconques selon l'algorithme EDF et donne les résultats du test de faisabilité suffisant :

	Ci	**Di**	**Pi**
τ1	3	7	20
τ2	2	4	5
τ3	1	8	10



Bien qu'aucune tâche ne dépasse son échéance, la condition suffisante n'est pas satisfaite :

$3/7 + 2/4 + 1/8 \leq 1$

$(24+28+7)/56 \leq 1$

$59/56 \leq 1$

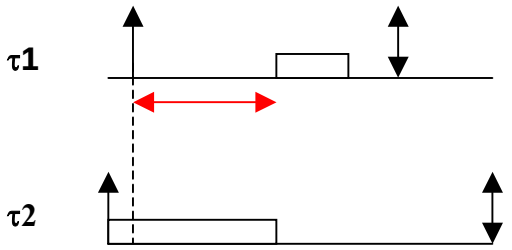
$1{,}05 \leq 1$ (test non respecté)

2.3.2) EDF non Préemptif

Nous présentons ici un test qui permet l'obtention des **conditions nécessaires et suffisantes** d'ordonnançabilité pour les tâches à échéance sur requête. Ce test est défini dans **[JSM91].** Par contre, aucun test pour l'algorithme EDF non préemptif avec des tâches quelconques n'a pu être trouvé.

Pour bien comprendre le test de faisabilité pour des tâches à échéance sur requête, il est nécessaire de rappeler ce qu'est **le pire temps de déphasage** dont nous avons déjà parlé à l'occasion du test de Rate Monotonic non préemptif. Ce pire temps de déphasage se produit lorsque la requête d'une tâche arrive une unité de temps avant l'arrivée des requêtes des taches plus prioritaire. Le phénomène est appelé : « inversion de priorité ».

Exemple : la tâche **τ1** dispose d'une priorité supérieure à la tâche **τ2** mais elle démarre un temps après la tâche **τ2**. Le pire temps de déphasage de la tâche **τ1** est représentée en rouge.



Le test de faisabilité va tenir compte de ce pire temps de déphasage. Il va vérifier que toutes les tâches de priorité supérieure ne surcharge pas le processeur sur une période de temps donnée.

Le test ne s'occupe que des tâches ayant leur échéance avant la fin de cette période donnée. Les tâches ayant leur échéance après cette période sont forcément moins prioritaires et ne doivent pas être prises en compte. Le test est représenté par deux conditions.

$$\sum_{i=1}^{n} Ci/Pi \leq 1$$

$$Ci + \sum_{j=1}^{i-1} \lfloor L-1/Pj \rfloor * Cj \leq L \quad \begin{cases} 1 < i \leq n \\ P1 < L < Pi \end{cases} \qquad (9)$$

- La première condition est une garantie de non surcharge du système

- La deuxième condition suppose que les tâches soient rangées par ordre d'échéance (de la plus petite à la plus grande). De plus la condition doit être vérifiée pour toutes les tâches.

La détermination de la condition s'effectue de la manière suivante :

L'intervalle de temps critique L est encadré tel que $P1+1 \leq L < Pi$. Cet intervalle est dû au fait que la tâche de priorité inférieure démarre un temps avant les autres tâches.

Il faut dans un premier temps, pour chaque tâche j plus prioritaire, déterminer le nombre de fois qu'elle est invoquée dans l'intervalle de temps L-1 (on tient compte du pire temps de déphasage). Ce nombre est multiplié par Cj afin de connaître le temps total de la tâche j dans cet intervalle.

A ces calculs on ajoute la capacité Ci de la tâche **τi** ce qui permet de connaître le temps maximum nécessaires aux tâches j dans un pire cas de déphasage. Cette valeur doit être comparée à l'intervalle de temps critique et ne doit pas le dépasser.

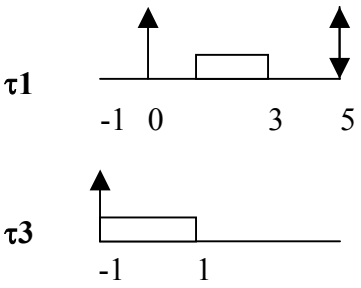
L'exemple suivant illustre le fonctionnement du test de faisabilité.

	Ci	**Di**	**Pi**
τ1	2	5	5
τ2	3	20	20
τ3	2	10	10

L'exemple présente le test de faisabilité sur la tâche **τ3**. Le temps critique L est situé entre P1 et P3 (non compris). C'est à dire : **5 < L < 10.** Seuls les calculs limites (L = 6 et L = 9) sont présentés.

- **L = P1 + 1 = 5 +1 =6**

La tâche **τ1** est bloquée par une tâche de priorité inférieure : **τ3**. Ici on ne tient pas compte de la tâche **τ2** car celle-ci dispose d'une échéance plus lointaine donc elle est moins prioritaire.



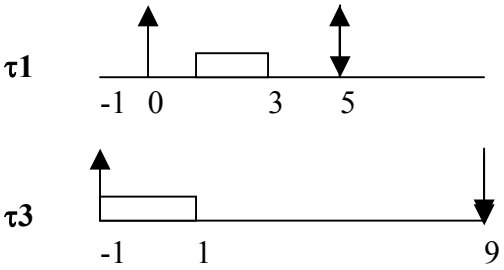
Pour L=6, la tâche **τ1** ne dépasse pas son délai critique.

$2 + \lfloor (6-1)/5 \rfloor * 2 \leq 6$

$4 \leq 6$ (test respecté)

- **L = P3 - 1 = 10 –1 =9**

On ne tient toujours pas compte de la tâche 2 car son échéance est hors de la période d'évaluation.



Pour L=9, la tâche **τ1** ne dépasse pas son délai critique.

$2 + \lfloor (9-1)/5 \rfloor * 2 \leq 9$

$4 \leq 9$ (test respecté)

Dans ce test, l'idée de priorité dynamique est respectée : la division entière par le bas permet au test de ne pas tenir compte des tâches dont la période n'est pas comprise dans le délai. Par contre ce test s'avère très fastidieux à utiliser car de nombreux calculs sont à effectuer pour chaque tâche. L'intérêt du test de faisabilité (vérification rapide qu'un ordonnancement est possible) s'en trouve fortement diminué.

2.4) Least Laxity First

C'est un algorithme basé sur des priorités dynamiques. La priorité d'une tâche est inversement proportionnelle à sa laxité dynamique. Autrement dit la tâche de plus haute priorité à l'instant t dispose de la plus petite laxité.

Les tests d'ordonnancabilité pour cet algorithme sont identiques aux tests de EDF dans le cas préemptif comme nous l'expliquons ci-après. Par contre, les tests de EDF non préemptif ne peuvent pas être utilisés pour LLF non préemptif.

Nous n'avons trouvé aucun test dans le cas non préemptif mais nous expliquons que la séquence d'ordonnancement n'est pas identique et que les tests de EDF ne peuvent pas être utilisés.

2.4.1) LLF préemptif

[CDK00] indique que les conditions d'ordonnançabilités pour Least Laxity First sont les mêmes que pour Earliest Deadline.

Pour des **tâches à échéances sur requête,** la **condition suffisante et nécessaire** est :

$$\sum_{i=1}^{n} Ci / Pi \leq 1$$

Pour les **tâches quelconques** la **condition suffisante** est :

$$\sum_{i=1}^{n} Ci / Di \leq 1$$

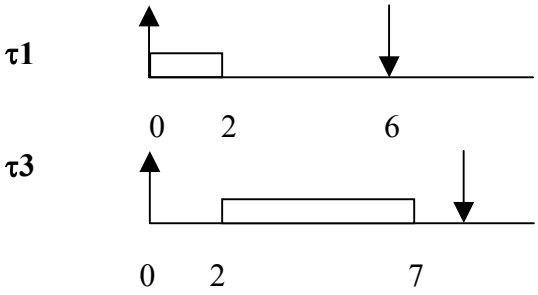
La **condition nécessaire** est celle donnée par la première formule.

2.4.2) LLF non préemptif

Contrairement au cas préemptif, les algorithmes LLF et EDF en non préemptif ne peuvent pas utiliser le même test de faisabilité. En effet, le test de EDF non préemptif vérifie qu'une tâche ne dépasse pas son échéance et l'ordonnancement de EDF et de LLF non préemptif s'avèrent différents dans certains cas. L'exemple suivant présente l'ordonnancement de deux tâches sous EDF puis sous LLF. On constatera que dans le cas de LLF, la première tâche dépasse son échéance. Ainsi LLF n'est pas optimal par rapport à EDF **[MOK83].**

	Ci	**Di**
τ1	2	6
τ3	5	8

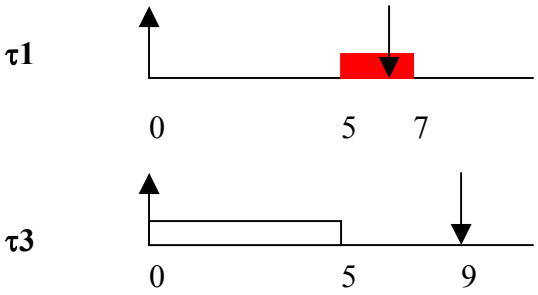
L’ordonnancement sous EDF non préemptif est le suivant :



D(0)=6-0=6

D(0)=8-0=8

L’ordonnancement sous LLF non préemptif est le suivant :



L(0)=D(0)-2=4

L(0)=D(0)-5=3

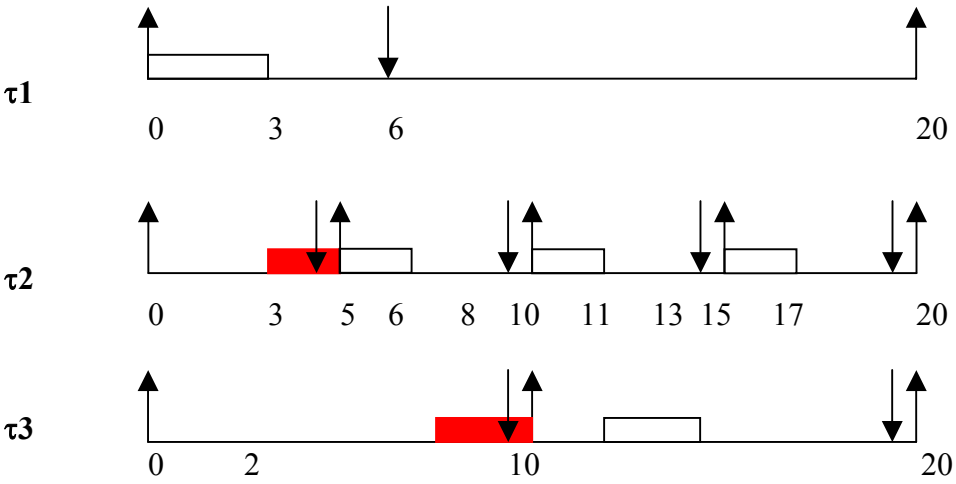
2.5) Fixed Preemptive Priority

Une **condition nécessaire et suffisante** est donnée dans **[LSD 89]**. Cette condition appelée TDA (Time Demand Analysis), utilise une formulation de la demande en ressource processeur pour chaque tâche. Elle est valable pour le cas simple ∀ i, Di ≤ Pi. avec affectation des priorités quelconques :

$$\forall\ 1 \le i \le n \quad \min_{0 \le t \le Di} \left(\sum_{j=1}^{i} Cj/t * \lceil t/Pj \rceil \right) \le 1 \qquad (10)$$

Dans ce test, les tâches doivent être ordonnées par ordre de priorité, c'est à dire en fonction de leur échéance (de la plus petite à la plus grande). Une utilisation de ce test est donnée dans l'exemple suivant.

	Ci	Di	Pi
τ1	3	7	20
τ2	2	4	5
τ3	2	9	10



Nous constatons sur le schéma que les tâche **τ2** et **τ3** dépassent leur échéance. En utilisant le test de faisabilité nous allons le vérifier sur la tâche **τ2**. Il faut alors calculer toutes les valeurs de **t** comprises entre 0 et D2 (c'est-à-dire entre 0 et 4) et prendre la valeur minimale trouvée.

Pour t=0

$C1/0 * \lceil 0/P1 \rceil + C2/0 * \lceil 1/P2 \rceil$
(division par zéro impossible)

Pour t=1

$C1/1 * \lceil 1/P1 \rceil + C2/1 * \lceil 1/P2 \rceil$
$3/1 * \lceil 1/20 \rceil + 2/1 * \lceil 1/5 \rceil$
$3*1 + 2 *1=5$

Pour t=2

$C1/2 * \lceil 2/P1 \rceil + C2/2 * \lceil 2/P2 \rceil$
$3/2 * \lceil 2/20 \rceil + 2/2 * \lceil 2/5 \rceil$
$3/2*1 + 2/2*1=5/2$

Pour t=3

$C1/3 * \lceil 3/P1 \rceil + C2/3 * \lceil 3/P2 \rceil$
$3/3 * \lceil 3/20 \rceil + 2/3 * \lceil 3/5 \rceil$
$3/3*1 + 2/3*1=5/32$

Pour t=4

$C1/4 * \lceil 4/P1 \rceil + C2/4 * \lceil 4/P2 \rceil$
$3/4 * \lceil 4/20 \rceil + 2/4* \lceil 4/5 \rceil$
$3/4*1 + 2/4*1=5/4$

Le minimum est 5/4 mais il est supérieur à 1 donc ce n'est pas ordonnançable.

Ce test donne une **condition nécessaire et suffisante** car il calcule au plus juste toutes les conditions d'ordonnancement qui peuvent se présenter. Toutefois, on constatera qu'une fois de plus, le nombre de calculs à effectuer reste important.

Mais il est également intéressant de souligner que ce test peut être utilisé pour les algorithmes RM et DM préemptif en ordonnant les tâches par priorité. Pour RM, l'ordre se fera en fonction des périodes car les échéances sont égales au périodes (Di=Pi). Pour RM comme pour DM, ce test s'avèrera plus judicieux que leurs tests respectifs.

2.6) Conclusion

Pour conclure cette première partie, nous pouvons dire que nos recherches n'ont pas été vaines car nous avons pu rapporter un bon nombre de tests de faisabilité. Ceux-ci sont présentés en résumé dans le tableau de la page suivante. Il est intéressant de souligner que peu de tests se révèlent à la fois nécessaires et suffisants. En effet, seuls les tests d'EDF préemptif pour des tâches à échéance sur requête et de FPP répondent à ces deux critères.

Certains tests faisant l'objet de notre recherche manquent à l'appel : DM non préemptif, LLF non préemptif, Non Fixed Preemptive Priority et Round Robin. Dans la partie suivante nous tentons de trouver des tests pour ces politiques.

Tableau récapitulatif des tests de faisabilité

	Preemptif	Non Preemptif
Rate Monotonic	$U = \sum_{i=1}^{n} Ci / Pi$ **S** $U \leq n * (2^{1/n} - 1)$	$\forall i \ 1 \leq i \leq n$ **S** $\sum_{j=1}^{i} Cj/Pj + Bi/Pi \leq i * (2^{1/i} - 1)$
Deadline Monotonic	$\sum_{i=1}^{n} Ci / Di \leq n * (2^{1/n} - 1)$ **S** $\forall i \ 1 \leq i \leq n$ **S** $Ci + \sum_{j=1}^{i-1} \lceil Di / Pj \rceil * Cj \leq Di$	*Aucune solution trouvée*
Earliest Deadline First	$\sum_{i=1}^{n} Ci / Pi \leq 1$ **NS** $\sum_{i=1}^{n} Ci / Di \leq 1$ **S**	$\sum_{i=1}^{n} Ci/Pi \leq 1$ **NS** $\begin{cases} 1 < i \leq n \\ P1 < L < Pi \end{cases}$ $Ci + \sum_{j=1}^{i-1} \lfloor L-1/Pj \rfloor \ Cj \leq L$ **Tâches quelconques** *Aucune solution trouvée*
Least Laxity First	**Tâches à échéances sur requête** $\sum_{i=1}^{n} Ci / Pi \leq 1$ **NS** **Tâches quelconques** $\sum_{i=1}^{n} Ci / Di \leq 1$ **S**	*Aucune solution trouvée*
Fixed Priority	$\forall \ 1 \leq i \leq n$ **NS** $\min_{0 \leq t \leq Di} (\sum_{j=1}^{i} Cj/t * \lceil t/Pj \rceil) \leq 1$	*Aucune solution trouvée*
Round Robin	*Aucune solution trouvée*	*Aucune solution trouvée*

Légende : **N** : Nécessaire **S** : Suffisant

3) Propositions et difficultés d'élaboration pour les tests manquants

Cette partie du rapport expose l'étude que nous avons mené sur les tests de faisabilité non trouvés lors de notre recherche.

En premier lieu, nous émettons des propositions de tests pour certains algorithmes (DM non préemptif, EDF non préemptif avec des tâches à échéance avant requête et Non Preemptive Fixed Priority).

En second lieu, nous montrons les difficultés rencontrées à l'élaboration d'un test pour les autres algorithmes (LLF non préemptif et Round Robin).

3.1) Les propositions

3.1.1) Deadline Monotonic non préemptif

Nos recherches sur un test de faisabilité pour Deadline Monotonic dans le cas non préemptif se sont avérées infructueuses. Ainsi nous proposons de reprendre les tests de Rate Monotonic non préemptif, vus précédemment, en remplaçant les P (périodes) par des D (échéances).

Cette réflexion a été menée en comparant les tests de RM*(1)* et de DM préemptifs*(5)*. En effet, le premier effectue la somme des Ci/Pi et le second la somme des Ci/Di. Comme RM non préemptif*(3)(4)* reprend le principe de RM préemptif mais en ajoutant un temps de blocage, il semble logique que DM non préemptif peut en être déduit en utilisant les échéances à la place des périodes.

1er test :

$$\forall i \ \ 1 \le i \le n \quad \sum_{j=1}^{i} Cj/Dj + Bi/Di \le i * (2^{1/i} - 1) \qquad (11)$$

2ème test:

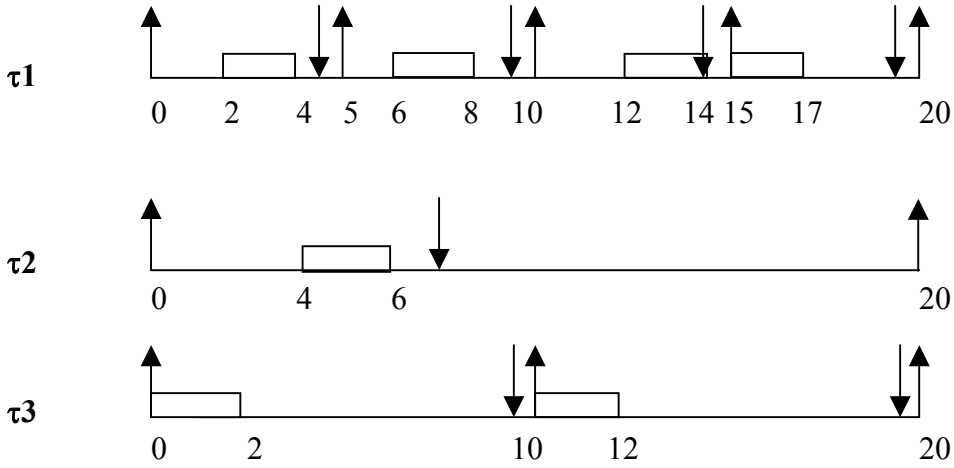
$$\sum_{i=1}^{n} Ci/Di + \max_{1 < i \le n} \{ Bi/Di \} \le n * (2^{1/n} - 1) \qquad (12)$$

Avec

$$Bi = \max_{i+1 \le j \le n} \{ Cj \}$$

L'exemple suivant illustre le 1er test de faisabilité pour une série de trois tâches ordonnancées selon Deadline Monotonic non préemptif. Ce test est uniquement effectué sur la tâche 2. Le schéma illustre le test de faisabilité et montre que les tâches 1 et 2 sont bloquées par une tâche de priorité inférieure

	Ci	**Di**	**Pi**
τ1	2	4	5
τ2	2	7	20
τ3	2	9	10



Le test de faisabilité n'est pas satisfait lorsqu'il est effectué sur la tâche 2 bien que le schéma est montré qu'aucune tâche ne dépasse sont échéance même en cas de blocage de la tâche 3. Ainsi, ce test est pessimiste.

$C1/D1 + C2/D2 + B2/D2 \leq 2 * (2^{1/2} - 1)$
$2/4 + 2/7 + 2/7 \leq 2 * (2^{1/2} - 1)$
$(14+8+8)/28 \leq 0,83$
$30/28 \leq 0,83$
$1,07 \leq 0,83$ (test non respecté)

3.1.2) EDF non préemptif avec des tâches à échéance avant requête

Nous proposons le test suivant pour les tâches à échéance avant requête :

$$\sum_{i=1}^{n} Ci/Pi \leq 1$$

$$Ci + \sum_{j=1}^{i-1} \lfloor L-1/Pj \rfloor * Cj + \sum_{j=1}^{i-1} Vj \leq L \qquad \begin{cases} 1 < i \leq n \\ D1 < L < Di \end{cases}$$

$$Vj \begin{cases} Cj & SI\ L - (Dj + \lfloor (L-1)/Pj \rfloor * Pj) > 0 \\ 0 & SINON \end{cases}$$

(13)

Ce test de faisabilité reprend le test d'EDF non préemptif avec des tâches à échéance sur requête*(9)* en y ajoutant la notion d'échéance. Ainsi ce test vérifie que chaque tâche ne dépasse pas son échéance grâce au calcul de Vj.

Le calcul de Vj repose sur l'idée de savoir si l'échéance d'une tâche j est située dans la période L-1.

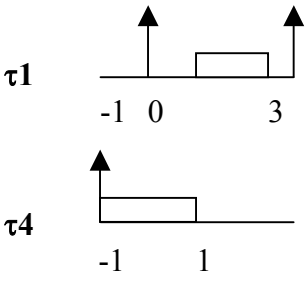
Pour ce faire on soustrait à la période L-1 d'une part le nombre d'activation de la tâche multiplié par sa période et d'autre part la valeur de son échéance. Si ce calcul est supérieur où égal à 0, cela signifie que pour la dernière activation de la tâche j, son échéance est située dans la période L-1.

L'exemple suivant illustre ce test pour la tâche **τ4**. Les tâches sont rangées par ordre d'échéance dans le tableau.

	Ci	Di	Pi
τ1	2	4	5
τ2	3	8	10
τ3	3	14	15
τ4	2	18	20

- **L = D1 + 1 = 4+1 =5**

La tâche **τ1** est bloquée par une tâche de priorité inférieure : **τ4**. Ici on ne tient pas compte des tâches **τ2** et **τ3** car celles-ci disposent d'une échéance supérieure donc elles sont moins prioritaires. Le calcul de V1, V2 et V3 permet de savoir si ces trois tâches ont l'échéance de leur dernière activation située dans la période L-1, c'est à dire 4.



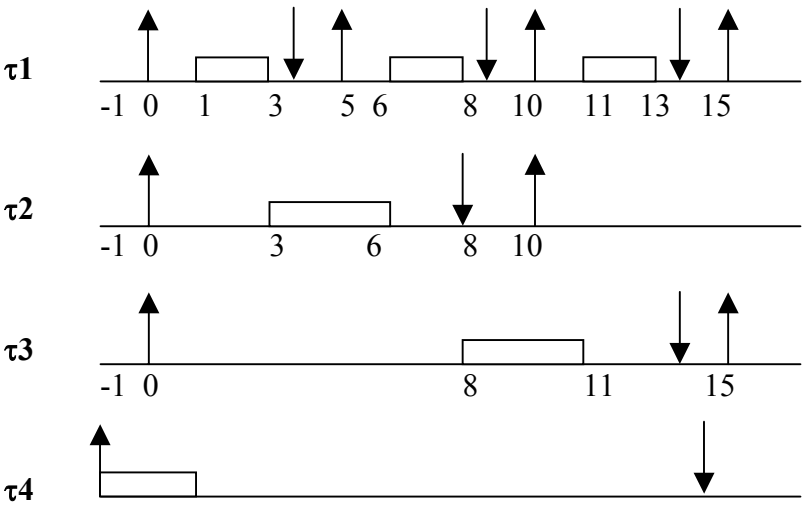
Pour L=5, la tâche **τ1** ne dépasse pas son délai critique.

L-1=5-1=4
V1=0 car 4- (4 + ⌊4/4 ⌋ * 5)= 4-(4+2)=4-6=-2 (<0)
V2=0 car 4- (8 +⌊4/8 ⌋ * 10)= 4-8=-4 (<0)
V3=0 car 4- (9 + ⌊4/14 ⌋ * 15)=4-9=-5 (<0)

C4 +⌊4/P1 ⌋* C1 + ⌊4/P2 ⌋*C2 +⌊4/P3 ⌋*C3 + V1 +V2 +V3 ≤ 5
2 +⌊ 4/4 ⌋* 2 + 0 ≤ 5
4 ≤ 5 (test respecté)

- **L = 15**

Le cas de L=15 illustre bien l'idée de vérification de l'échéance pour chaque tâche.



-1 1

Le calcul de V1 retourne 2 car l'échéance de la dernière activation de la tâche 1est située dans la période. Il en est de même pour la tâche 3, ainsi V3 = 3.

L-1=15-1=14
V1=2 car V1=0 car 14- (4 + ⌊14/5 ⌋* 5)= 14-(4+10)=14-14=0 (≥0)
V2=0 car 14- (8 +⌊14/10 ⌋* 10)= 14-18=-4 (<0)
V3=3 car 4- (14 + ⌊14/15 ⌋* 15)=14-14=0 (≥0)

C4 +⌊14/P1 ⌋* C1 + ⌊14/P2 ⌋*C2 +⌊14/P3 ⌋*C3 + V1 +V2 +V3 ≤ 15
2 +⌊14/5 ⌋* 2 +⌊14/10 ⌋* 3 + 5 ≤ 14
2+ 4+3+5 ≤ 15
14 ≤ 15 (test respecté)

3.2.1) Non Preemptive Fixed Priority

Nous avons établi une condition de faisabilité pessimiste pour Non Preemptive Fixed Priority.

$$\forall\ 1 \le i \le n \qquad Ci + \sum_{j=1}^{i-1} Cj * \lceil Di/Pj \rceil + Bi \le Di \qquad (14)$$

Avec $Bi = \max_{i+1 \le j \le n} (Cj) - 1$

Ce test de faisabilité s'inspire du test de Fixed Preemptive Priority*(10)*. Il suppose que les tâches soient ordonnées selon priorité. Pour chaque tâche on cherche à vérifier que dans le pire cas de déphasage sa capacité ne dépasse pas son échéance.

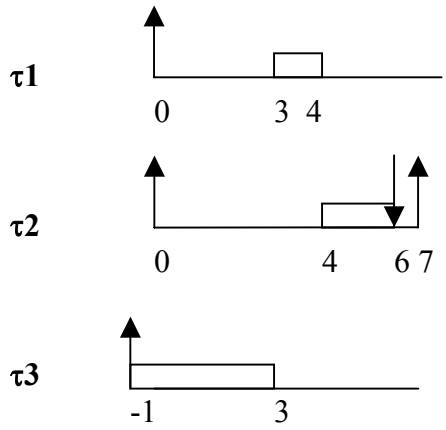
La partie gauche du test se calcule en tenant compte de la capacité de la tâche, des tâches plus prioritaire sur la même période et de la tâche moins prioritaire disposant de la plus grande capacité (Bi).

La division par le haut veut dire que la formule donne **toujours** la priorité à une tâche disposant d'une priorité plus forte. Il ne faut pas oublier que les tâches sont rangées par ordre de priorité.

L'exemple illustre l'utilisation du test de faisabilité sur une série de trois tâches avec priorité quelconque. Le test est appliqué uniquement à la tâche 2.

	Priorité	**Ci**	**Di**	**Pi**
τ1	1	1	9	10
τ2	2	2	6	7
τ3	3	4	7	20

Le graphique ci-dessous présente la manière dont le calcul du test de faisabilité est effectué.



$B2 = \max_{3 \le j \le 3} (Cj) - 1 = 4-1 = 3$

$C2 + C1 * \lceil D2/P1 \rceil + B2 \le D2$

$2 + 1 * \lceil 6/10 \rceil + 3 \le 6$

$2 + 1 * 1 + 3 \le 6$

$6 \le 6$ (test respecté)

3.2) Les difficultés d'élaboration pour les autres algorithmes

3.2.1) LLF non préemptif

Aucun algorithme pour LLF non préemptif n'a pu être trouvé. Nous avons expliqué que dans le cas non préemptif, les tests de EDF non préemptif ne peuvent pas être utilisés. Nous n'avons pas réussi à élaborer de test suffisamment satisfaisant. La difficulté réside dans la complexité du calcul de la priorité (la laxité).

3.2.2) Round Robin

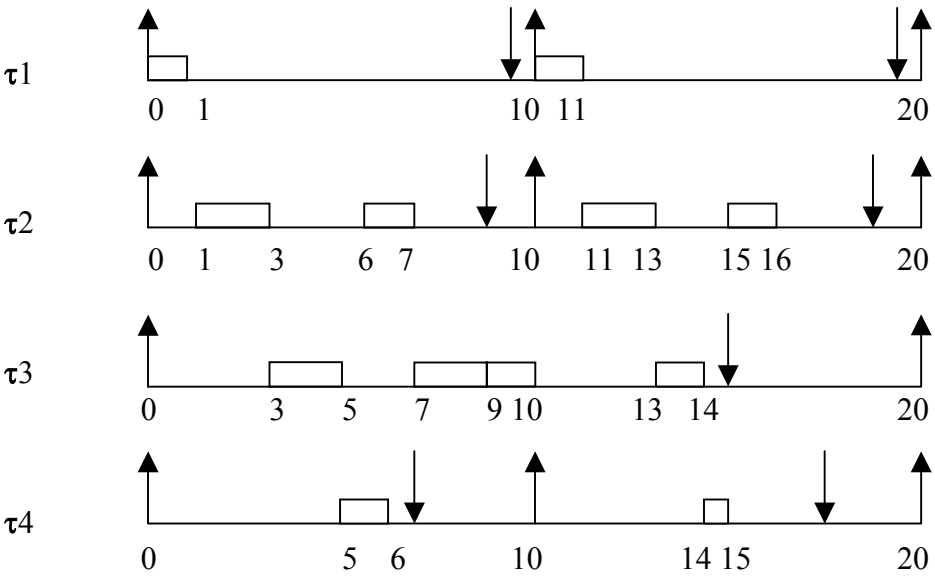
Le grand absent dans ce rapport est le test pour la politique Round Robin. En effet, nous n'avons trouvé aucun test de faisabilité. Nous pouvons tout de même signaler que nos recherches nous on conduit à trouver des calculs de temps de réponse sur la politique Round Robin **[MAL99].**
Comme aucun test n'a pu être trouvé, nous avons essayé d'en réaliser un.

Commençons tout d'abord par le cas non préemptif. Nous avons remarqué que Round Robin en non préemptif équivaut à FPP. Dans ce cas, le quantum n'a plus d'utilité. Cela signifie qu'il n'y a plus de préemption et qu'il s'agit bien d'une politique FPP.

Round Robin en préemptif, c'est à dire avec l'utilisation d'un quantum pose de nombreux problèmes quant à la réalisation d'un test de faisabilité. Nous expliquons au travers d'un exemple les difficultés que nous avons rencontré. Dans cet exemple le quantum vaut 2, la tâche 1 dispose de la priorité la plus forte, les tâches 2, 3 et 4 sont regroupées dans une seule FIFO avec une priorité inférieure.

	Priorité	**Ci**	**Di**	**Pi**
τ1	**1**	1	9	10
τ2	**2**	3	8	10
τ3	**2**	6	15	20
τ4	**2**	1	7	10

La série de tâches s'ordonnance de la manière suivante :



Afin de chercher un test de faisabilité, il nous a semblé plus simple de modéliser les tâches de façon à les regrouper par FIFO. La modélisation que nous avons envisagé est la suivante :

Une FIFO dispose d'un numéro de priorité, la FIFO 1 étant la plus prioritaire. Chaque tâche est reconnaissable par un couple numéro FIFO, numéro tâche (f,t). Le numéro de tâche étant le numéro d'ordre dans la FIFO. Pour l'exemple la modélisation devient la suivante :

	Modélisation	**C(f,t)**	**D(f,t)**	**P(f,t)**
τ1	**τ(1,1)**	1	9	10
τ2	**τ(2,1)**	3	8	10
τ3	**τ(2,2)**	6	15	20
τ4	**τ(2,3)**	1	7	10

Nous avons essayé d'effectué un test sur chaque tâche en vérifiant que sa capacité ne dépasse pas son échéance. Mais de nombreux cas de figures empêchent de créer un test suffisamment réaliste.

Tout d'abord, toutes les tâches d'une FIFO f sont contraintes par les tâches situées dans les FIFOS de priorité supérieure (1 à f-1). Le temps d'attente d'une tâche **τ** (f,t) située dans une FIFO f avant de pouvoir commencer à se réaliser est le suivant :

Temps total d'une file k sur une période D(f,t)

$$Temps\_File(k) = \sum_{j=1}^{n_k} C(j,k) * \lceil D(f,t)/P(j,k) \rceil$$

Temps total de toutes le files situées avant la file f

$$Total\_Files(f) = \sum_{k=1}^{f-1} Temps\_File(k)$$

(15)

Par exemple Pour la tâche **τ(2,1)**,c'est à dire la tâche **τ2** le temps d'attente avant de pouvoir démarrer est le suivant :

$$Temps\_File(1) = \sum_{j=1}^{1} C(j,1) * \lceil D(2,1)/P(j,1) \rceil$$

$Temps\_File(1) = C(1,1) * \lceil D(2,1)/P(1,1) \rceil$
$Temps\_File(1) = 1 * \lceil 8 / 10 \rceil = 1*1$
$Temps\_File(1) = 1$
$Total\_Files(2) = Temps\_File(1) = 1$

Le problème le plus important apparaît quand une tâche appartenant à une file est contrainte par les autres tâches situées avant et après elle dans la file.

On a essayé d'effectuer un test sur la première tâche dans une file, comme par exemple la tâche t2, A chaque préemption, la tâche doit attendre que toutes les autres tâches aient réalisé leur quantum avant de continuer. Il est possible de déterminer l'attente maximale d'une tâche (f,t) entre deux préemptions.

Pour toutes tâches t appartenant à une file f

$$\text{Blocage}(f,t) = \sum_{j=1}^{n_f} \min(\text{Quantum}, C(f,j)) \quad \text{avec } j \neq i$$

n_f = nombre de tâches de la fifo f

(16)

Le Blocage maximum de la tâche t(2,1) est :
Blocage(f,t)= Min (2,6) + Min (1,6) = 2 +1 = 3

Il est également possible de calculer le nombre de préemption d'une tâche t appartenant à une file f.

$$\text{Nb\_Préemption}(f,t) = \lfloor Ci/\text{Quantum} \rfloor \qquad (17)$$

Par exemple pour la tâche (2,1):
Nb_Préemption(2,1)=$\lceil 3/2 \rceil = 1$

Ainsi on connaît le nombre de préemption de la tâche i et le temps maximal de blocage de cette tâche entre deux préemptions. Un test pessimiste peut consister à vérifier si une tâche ne dépasse pas son délai en fonction de la capacité de la tâche, du nombre de préemption et du temps maximal de blocage :

$$\text{Total\_Files}(f) + \text{Blocage}(f,t) * \text{Nb\_Préemption}(f,t) + C(f,t) \leq D(f,t) \qquad (18)$$

Pour la tâche 2 cela donne le test résultat suivant :
Total_Files(2) + Blocages(2,1) * Nb_Préemption(2,1) + C(2,1) ≤ D(2,1)
1 + 3 * 1 ≤ 8
4 ≤ 8

Malheureusement ce test n'est pas réaliste car beaucoup trop pessimiste. En effet, le test se base sur le fait qu'à chaque préemption, une tâche doit attendre le temps que toutes les tâches utilisent leur quantum. Mais, il arrive bien souvent que ces tâches aient complètement fini de s'exécuter où n'utilise pas tout leur quantum de temps. Par exemple, à chaque préemption de la tâche $\tau 3$, celle-ci n'a pas besoin d'attendre le temps maximal d'attente. Ce sont les difficultés de réaliser au plus juste les temps d'attente à chaque préemption qui nous ont bloqués dans la réalisation du test.

Il existe une piste que nous n'avons pas exploitée. C'est celle d'établir un test de faisablité en utilisant le temps de réponse d'une tâche à l'image du test existant pour RM préemptif. En effet, nous avons signalé que des études ont été faites sur les temps de réponse des tâches utilisant la politique Round Robin. Ainsi, il est possible d'utiliser ces calculs afin d'établir un test par contre il est fort possible que les calculs des temps de réponse engendre une complexité qui dénature l'idée de test de faisabilité rapide.

3.3) Conclusion

Quelques tests manquants à notre recherche ont été proposés dans cette partie du rapport. Ceux-ci sont dérivés des tests trouvés précédemment et adaptés pour de nouvelles situations. Deadline Monotonic non préemptif est dérivé de Rate Monotonic non préemptif. EDF non préemptif pour les tâches à échéance avant requête est dérivé de EDF non préemptif pour des tâches à échéance sur requête. Non Preemptive Fixed Priority s'inspire de Fixed Preemptive Prioriity et de RM non préemptif . On notera que le test proposé pour EDF non préemptif est difficile à utiliser à cause du nombre de calculs important qu'il engendre.

Nous avons également signalé qu'un test pour LLF non préemptif est difficile à réaliser à cause du calcul de la laxité qui sert de priorité aux tâches.

Dans le cas de Round Robin, nous avons tout d'abord expliqué que Round Robin non préemptif équivaut à FPP. Puis, nous avons montré les difficultés d'élaboration d'un test pour RR préemptif. Finalement, nous donnons une piste pour l'élaboration d'un test : l'utilisation des temps de réponse.

4) Le logiciel Cheddar

Comme nous l'avons mentionné, le logiciel freeware CHEDDAR permet une simulation et une vérification de l'ordonnancement d'un jeu de taches sous une politique donnée et des contraintes temps réel bien précises.

Ce logiciel a été développé sous Ada, par l'équipe EA2215 à l'Université de Bretagne Occidentale (Brest). Le logiciel est composé de deux parties indépendantes : une partie graphique qui représente la simulation d'un ordonnancement de tâches et une partie texte qui indique les résultats de différents calculs de faisabilité.

Il s'agissait pour nous d'implanter les tests de faisabilité pour les politiques RM*(1)*, DM*(5)*, EDF*(8)*, LLF*(8)* préemptives et RM*(3)*, EDF*(9)* non préemptives. Seuls deux cas n'auront pas été traités : DM et LLF non préemptif car aucun test n'a été trouvé. Le résultat des tests devait pouvoir être consulté dans la partie basse du logiciel sous forme d'une phrase indiquant le résultat du test de faisabilité.

Dans un premier temps, nous aborderons l'implémentation en Ada des tests pour les politiques préemptives et non préemptives. Dans un second temps, nous donnerons quelques exemples d'utilisation du logiciel Cheddar.

4.1) Implémentation des tests connus

L'organisation du code des ordonnanceurs est définit de la manière suivante. Un package de haut niveau appelé `scheduler` représente le squelette des ordonnanceurs. Deux autres packages (sous classes de `scheduler`) permettent de séparer les ordonnanceurs selon qu'il soit à priorité fixe (package `scheduler.Fixed_Priority`) ou dynamique (package `Scheduler.Dynamic_Priority`). Au plus bas niveau, on trouve les ordonnanceurs RM, DM et HPF qui sont des sous classes du package `scheduler.Fixed_Priority` et d'un autre côté EDF et LLF qui sont des sous classes du package `Scheduler.Dynamic_Priority`.

Nous avons implémenté une procédure `utilization_factor_feasibility_test` qui est générique à tous ces packages. C'est-à-dire qu'elle est définie au niveau le plus haut comme une procédure du package `scheduler` puis redéfinie dans chaque package représentant les ordonnanceurs.

La procédure `Compute_Feasibility_Processor_Utilization_Factor` est appelée au moment où l'utilisateur demande à calculer les taux de faisabilité. Cette procédure a uniquement besoin de contenir la procédure `utilization_factor_feasibility_test` pour tous les ordonnanceurs.

Pour chaque ordonnanceur la fonction traite le cas préemptif et non préemptif. Afin de savoir de quel cas il s'agit, il est nécessaire d'utiliser la fonction `Get_Preemptive()`.

L'implémentation des politiques préemptives a été facile car il existait déjà trois fonction utile au test :.

- La fonction `Bound_On_Processor_Utilization`, qui permet de calculer automatiquement la borne en fonction de la politique choisie était déjà créée.
- La fonction `Processor_Utilization_Over_Deadline`, qui permet de retourner la somme des Ci/Pi, existait également.
- La fonction `Processor_Utilization_Over_Period` sert à retourner le calcul des Ci/Di.

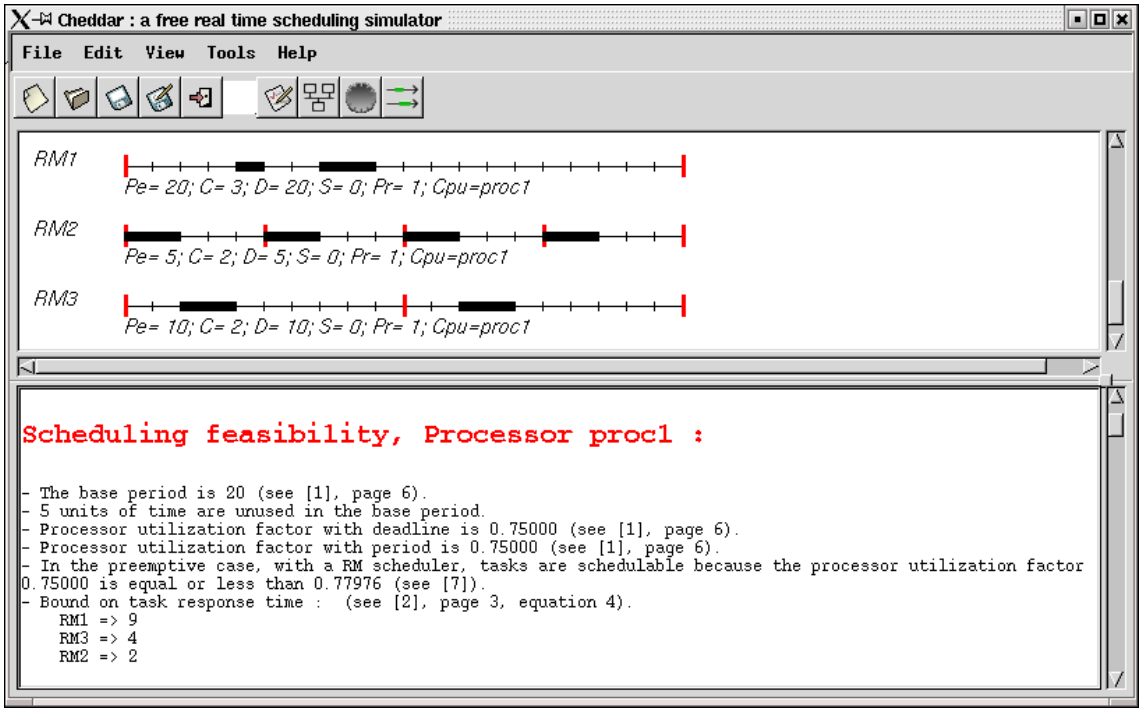
Pour chaque politique, il suffisait d'utiliser le résultat de ces fonctions afin d'effectuer le test de faisabilité. Par exemple, pour la politique EDF*(8)*, il a suffit d'effectuer la comparaison entre le résultat des Ci/Pi et de la borne.

Seuls les tests de faisabilité pour les politiques RM*(3)* et EDF*(9)* ont été implémentés pour le cas non préemptif. Dans les deux cas, il était nécessaire d'effectuer un test sur toutes les tâches préalablement triées selon leur ordre de priorité. Il est à signaler que même si un test pour une tâche ne se révèle pas satisfaisant, ce test sera tout de même effectué sur toutes les tâches suivantes. Ceci permet de récupérer et d'afficher le nom de toutes les tâches qui n'ont pas satisfait au test.

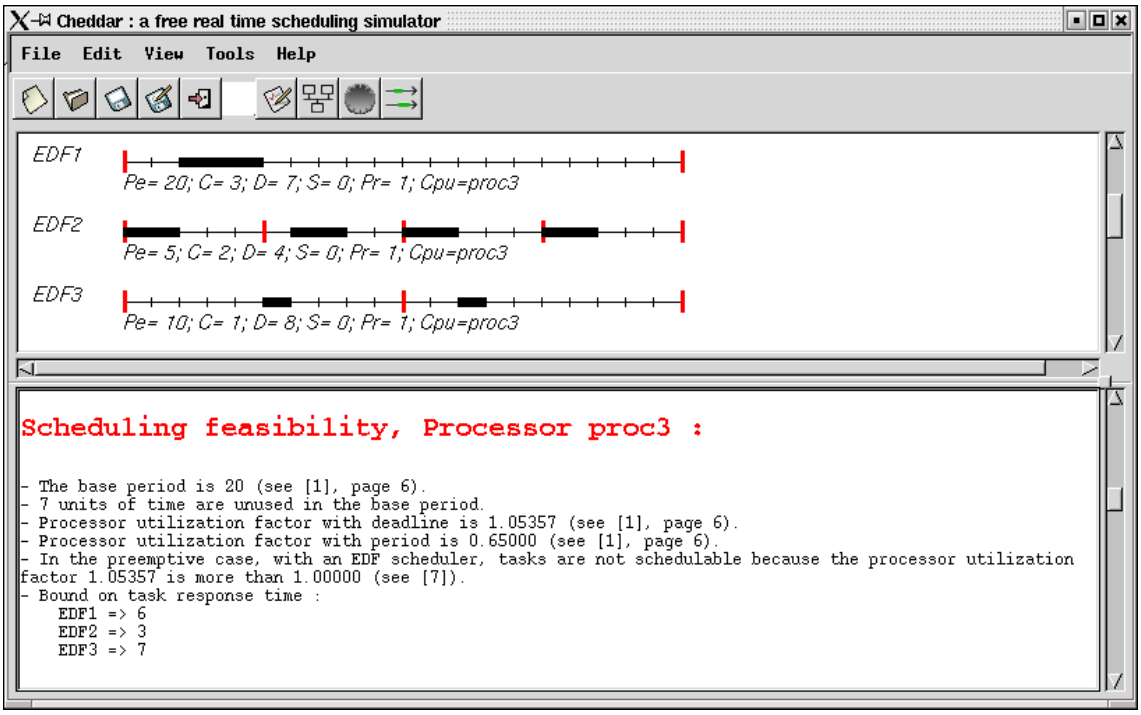
Une fois le test effectué, une ou plusieurs phrases doivent êtres inscrites à l'écran indiquant si la série de tâche à satisfait ou non au test. Ce texte doit s'afficher en français ou en anglais selon la langue qui a été choisi au moment de l'ouverture du logiciel CHEDDAR. Dans ce but, on utilise une seule variable pour retourner le texte en anglais ou en français. Cette variable dispose d'un paramètre qui peut avoir deux valeurs (Français ou English). Selon la valeur fournit en paramètre de la variable, la phrase retourner sera en anglais ou en français.

4.2) Jeux d'essai

Dans le cas préemptif, nous avons saisi 4 processeurs chacun sous une politique différente (RM, DM, EDF, LLF). Chaque processeur dispose de trois tâches. Nous présentons des copies d'écran pour les politiques RM et EDF. Le logiciel est divisé en deux parties. La partie haute présente une simulation de l'ordonnancement. La partie basse donne le résultat des tests de faisabilité.

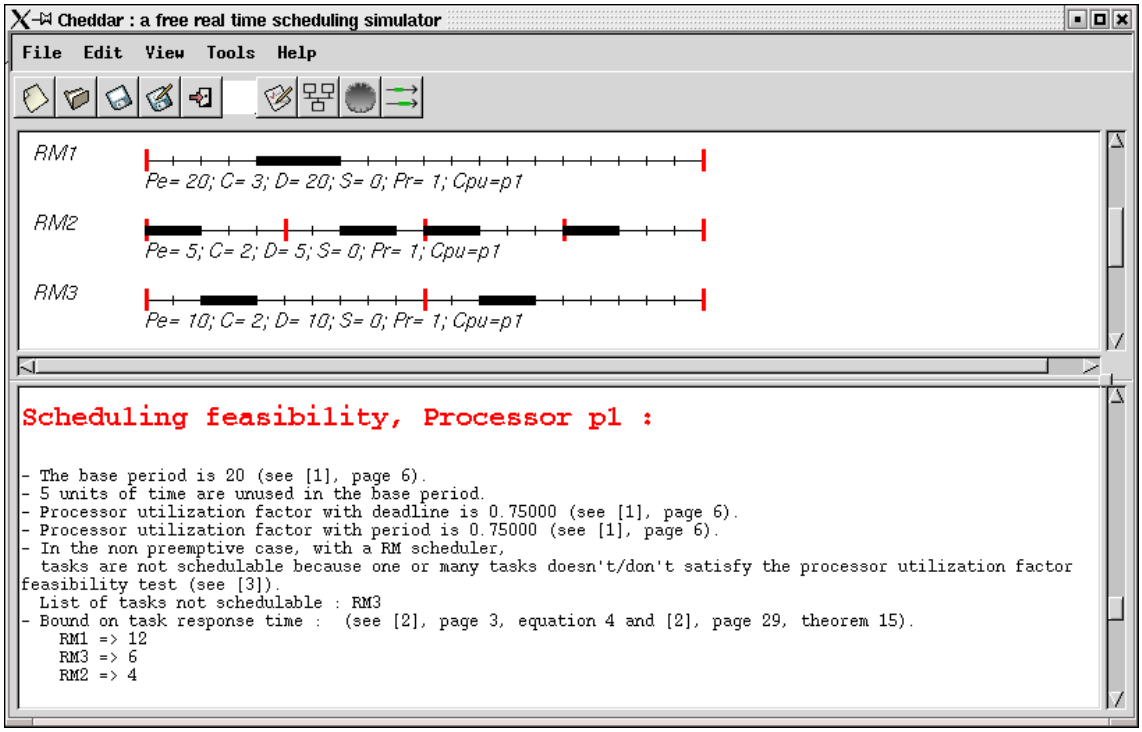


Pour l'exemple de RM, la simulation présente un ordonnancement où aucune tâche ne dépasse les bornes. Le test de faisabilité situé dans la partie basse à la 4ème ligne confirme ce résultat.

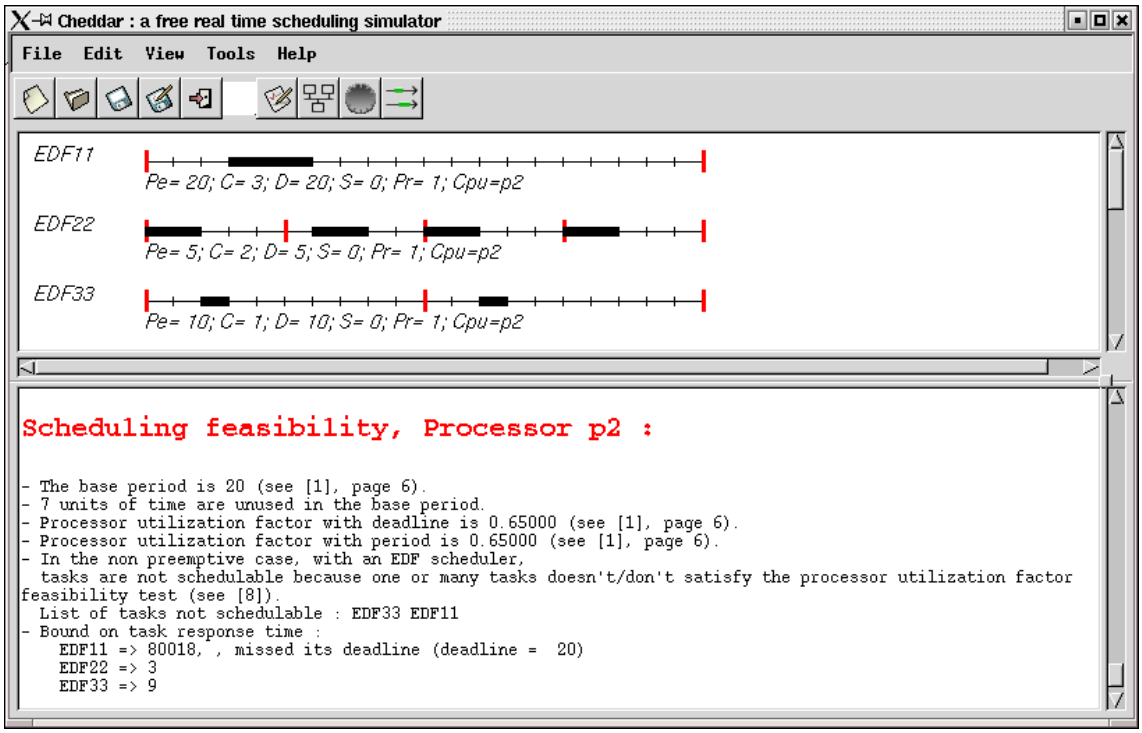


Pour l'exemple d'EDF, la simulation montre un ordonnancement sans qu'aucune tâche ne dépasse son échéance. Par contre, le test de faisabilité pessimiste indique que le taux d'utilisation dépasse la borne 1.

Deux tests ont été implantés dans le cas non préemptif. Il s'agit des tests de RM et de EDF pour tâches à échéance sur requête. Nous présentons ci-après les photos d'écran illustrant ces deux politiques.



pour RM non préemptif, la partie basse du logiciel écran montre que le test d'ordonnancement n'est pas satisfait à cause de la tâche RM2. Par contre la simulation avait montré un ordonnancement possible.



Le test d'EDF non préemptif signale les tâches qui ne satisfont pas au test : EDF33 et EDF 11.

4.3) Conclusion

Cette partie du rapport a résumé l'implémentation des tests de faisabilité pour les algorithmes préemptifs et non préemptifs. Cette implémentation a été effectué dans le logiciel CHEDDAR. Toutefois, il reste encore des tests à ajouter tel que DM non préemptif, LLF non préemptif et HPF.

5) Conclusion

Ce rapport a tout d'abord présenté une synthèse des différents tests de faisabilité existants (basé sur le temps processeur) pour les algorithmes principaux en préemptif et non préemptif. Nous avons pu collecter des tests pour les algorithmes RM, DM, EDF, LLF, FP préemptif, RM non préemptif et enfin pour EDF non préemptif à échéance sur requête. Les tests de EDF non préemptif et de FPP se révèlent assez contraignants à utiliser dû au nombre de calculs important qu'ils engendrent. Par ailleurs, nous avons implémenté la plupart des tests trouvés dans le logiciel CHEDDAR. Ce dernier a été créé par une équipe de l'Université de Bretagne Occidentale.

Notre recherche ne nous a pas permis de trouver tous les tests visés au départ. Nous avons alors suggéré des tests pour les algorithmes DM non préemptif, EDF non préemptif à échéance avant requête et NPFP. Ceux-ci reprennent certains tests que nous avons trouvés en les adaptant. Pour les politiques Round Robin et LLF non préemptif, nous n'avons pas pu établir de test qui nous semble satisfaisant. Pour Round Robin préemptif nous proposons tout de même d'orienter les recherches sur le calcul des temps de réponse car des recherches dans ce domaine ont été faites par L'INRIA **[MAL99]**. Toutefois, il faut vérifier si ces calculs ne sont pas trop complexes.

Pour conclure, il est important de souligner qu'il existe encore des travaux à effectuer dans ce domaine. En effet, d'une part très peu de tests s'avèrent nécessaires et suffisants en même temps. Ainsi, il est toujours possible d'améliorer les tests uniquement suffisants. Et d'autres part, pour certaines politiques nous n'avons trouvé aucun test basé sur le temps processeur. Des recherches peuvent être poursuivies afin de vérifier si des tests existent. Il est également possible de créer les tests en se basant éventuellement sur les propositions que nous avons faites.

Symboles et abréviations

τi : tâche i

Ci : Temps d'exécution de la tâche i

Pi : Période de la tâche i

Di : Délai critique de la tâche i

Bi : Temps de blocage de la tâche i

L(t) : Laxité résiduelle

D(t) : Délai critique résiduelle

RTi : Temps de réponse

U : facteur d'utilisation. Pour toutes les politiques d'ordonnancement $U = \sum_{i=1}^{n} Ci / Pi$

RM : Rate Monotonic

DM: Deadline Monotonic

EDF : Earliest First Deadline

LLF : Least Laxity First

FPP : Fixed Priority First

NPFP: Non Preemptive Fixed Priority

HPF : Highest Priority First

Références

[AB90] N. Audsley, A. Burns ; Real time Systems Scheduling ; YCS 134, Department of Computer Science, University of York, 1990

[CAR96] Francisco Vasques de Carvahlo ; Integration de Mecanismes d'Ordonnancement et de Communication dans la sous-Couche MAC de Reseaux Locaux Temps réel ; Thèse, Laboratoire d'Analyse et d'Architectures des Systèmes du CNRS, 1996 (N° ordre: 2340)

[CDK00] Francis Cottet, Joëlle Delacroix, Claude Kaiser, Zoubir Mammeri ; Ordonnancement temps réel ; Edition kermes Science, Janvier 2000

[DEC02] David Decotigny ; Bibliographie d'introduction à l'ordonnancement dans les systèmes informatiques temps réel ; Novembre 2002

[GRS96] L.George, N.Rivierre, M.Spuri ; Preemptive and non preemptive real time Uniprocessor Scheduling ; n°2966 - Sept. 1996; INRIA

[JP86] M. Joseph, P Pandaya ; Finding Response Times in real time System ; The Computer Journal 29(5), pp. 390-395, 1986

[JSM91] 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

[LL73] C.Liu & J. Layland ; Scheduling Algorithms for multiprogramming in a hard Real time environment ; Journal of ACM, vol n°29, n°1, 1973, pp 46-61

[LM80] J. Leung, M. Merril ; A Note on Preemptive Scheduling of Periodic Real-Time Tasks ; Information Processing Letters, 11 (3), pp 115-118, November 1980

[LSD89] J. Lehoczky, L. Sha, Y. Ding ; The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour ; IEEE RTS'89, pages 166-171, December 1989

[MAL99] Jörn Martin Migge, Alain Jean-Marie ; Real-Time Scheduling: Non-Preemption, Critical Sections and Round Robin ; Avril 1999 N° 3678

[MOK83] A.K. Mok ; Fundamental design Problems of distributed systems for the hard real time environment ; PhD thesis, Massachusetts Institute of Technology, Jun 1983

[SRL90] L. Sha, R.Rajkumar, J.Lehoczky ; Priority Inheritance Protocols; an approach to Real time Synchronization ; IEE Tr. on computers, 39(9), September 1990