

## **Première partie**

# **Contributions et Implementation**



# Chapitre 1

## Endormissement de tâche sous priorité Fixe

### Sommaire

---

1.1	Introduction . . . . .	12
1.2	Taxonomie sur les systèmes temps réel . . . . .	12
1.3	Modélisation des tâches . . . . .	12
1.4	Ordonnancement monoprocesseur . . . . .	14
1.4.1	Algorithme d'ordonnancement à priorité fixe . . . . .	14
	Rate Monotonic [?] . . . . .	14
	Deadline Monotonic [?] . . . . .	15
1.4.2	Algorithme d'ordonnancement à priorité dynamique . . . . .	16
	Earliest Deadline First[?] . . . . .	16
1.5	Ordonnancement Multiprocesseur . . . . .	18
1.5.1	Classification . . . . .	18
1.5.2	Optimalité . . . . .	18
1.5.3	Algorithme d'ordonnancement utilisant une stratégie par partitionnement .	19
	Généralité . . . . .	19
	First-Fit et Best-Fit . . . . .	19
1.6	Conclusion . . . . .	19

---

## 1.1 modèle de tâches

Le modèle utilisé ici est le modèle de tâche périodique de Liu et Layland défini au chapitre 1.

Soit  $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$  un ensemble de tâche, chaque tâche  $\tau_i$  est caractérisé par  $\tau_i = (C_i, D_i, T_i)$  et :

- $\tau_i$  est périodique.
- $C_i$  est le pire temps d'exécution de la tâche  $\tau_i$ .
- $D_i$  est l'échéance relative de la tâche  $\tau_i$ .
- $T_i$  est la période relative de la tâche  $\tau_i$ .
- l'ensemble de tâches  $\Gamma$  est ordonnançable avec l'algorithme Deadline Monotonic.

## 1.2 Le cas monoprocesseur

Dans cette section nous allons insérer une tâche d'endormissement  $\tau_{sleep} = \{C_{sleep}, D_{sleep}, T_{sleep}\}$  avec  $C_{sleepMin} \leq C_{sleep} \leq C_{sleepMax}$  avec  $C_{sleepMax} = (1 - U) \times T_H$  et  $\tau_{sleep} \cup \Gamma$  est ordonnançable avec Deadline Monotonic.

Pour cela nous présentons l'algorithme d'insertion de tâche d'endormissement  $\tau_{sleep}$  dans un taskset  $\Gamma$ .

---

### Algorithme 1 : Insertion Taches Endormissement Dans Un Mono-processeur

---

**Données :** TaskSet  $\Gamma$ , Temps Minimum d'exécution  $C_{SleepMin}$ , Temps d'exécution Maximum  $C_{SleepMax}$ , Pas de decrementation  $\Delta C$

**Résultat :** Tache  $\tau_{Sleep}$

```

1 début
2    $C_{Sleep} \leftarrow C_{SleepMax}$ ;
3    $D_{Sleep} \leftarrow T_H$ ;
4    $T_{Sleep} \leftarrow T_H$ ;
5   tant que  $\Gamma \cup \text{tache}(C_{Sleep}, D_{Sleep}, T_H)$  est non ordonnançable et  $C_{Sleep} \geq C_{SleepMin}$ 
6     faire
7        $C_{Sleep} \leftarrow C_{Sleep} - \Delta C$ ;
8   fin
9    $\tau_{Sleep} \leftarrow \text{CreerTache}(C_{Sleep}, D_{Sleep}, T_{Sleep})$ 
10 fin
```

---

Nous illustrons notre algorithme avec un exemple d'application. Le tableau 1.1 et la figure ?? représente un ensemble de tâches  $\Gamma$  ordonnançable par Deadline Monotonic où on a inséré une tâche d'endormissement  $\tau_{sleep}$

$\tau_i$	$C_i$	$D_i$	$T_i$
1	1	10	10
2	2	15	15

TABLE 1.1 – Ensemble de taches periodiques

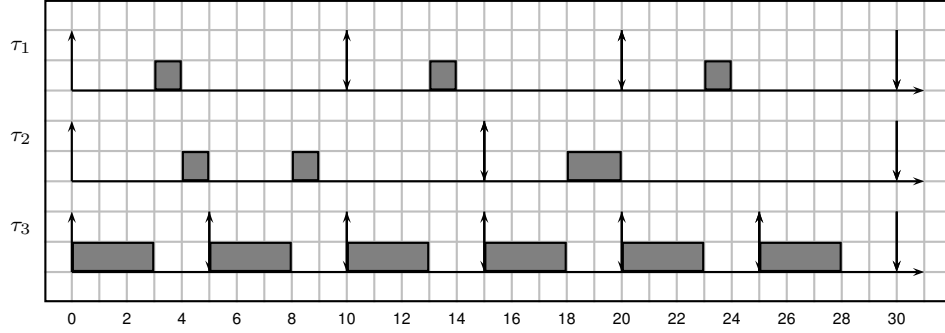


FIGURE 1.1 – Insertion de tache dans un monoprocesseur

### 1.3 Le cas multiprocesseur

Dans cette section nous allons inserer un ensemble de tâches d'endormissement  $\{\tau_{sleep}^1, \tau_{sleep}^2, \dots, \tau_{sleep}^m\}$  tel que  $\tau_{sleep}^i = \{C_{sleep}^i, D_{sleep}^i, T_{sleep}^i\}$  dans  $m$  processeurs  $P = \{P_1, P_2, \dots, P_m\}$ . Pour cela nous presentons nous presentons deux strategie d'insertion :

Insertion Locale : Chaque processeur  $i$  à sa propre tache d'endormissement  $\tau_{sleep}^i$

Insertion Globale : Tous les processeur ont une même tache d'endormissement  $\tau_{sleep} = \tau_{sleep}^1 = \tau_{sleep}^2 = \dots = \tau_{sleep}^m$

Les deux algorithmes representent l'insertion locale (Resp. globale) d'un ensemble de tâches d'endormissement dans un ensemble de processeur.

---

**Algorithme 2** : Insertion Globale de Taches Endormissement Dans Une architecture Multiprocesseur

---

**Données** : Ensemble de processeur  $P$ , Temps Minimum d'execution  $C_{SleepMin}$

**Résultat** : Ensemble de Tache d'endormissement  $\Gamma_{Sleep}$

```

1 début
2   pour  $P_i \in P$  faire
3      $\tau_{sleep}^i = \text{tache\_endormissement\_monoprocesseur}(C_{SleepMin});$ 
4      $\Gamma_{Sleep}^i = \Gamma_{Sleep}^i \cup \tau_{sleep}^i$ 
5   fin
6 fin
```

---

---

**Algorithme 3** : Insertion Globale de Taches Endormissement Dans Une architecture Multipro-  
cesseur
 

---

**Données** : Ensemble de processeur  $P$ , Temps Minimum d'exécution  $C_{SleepMin}$ 
**Résultat** : Ensemble de Tache d'endormissement  $\Gamma_{Sleep}$ 

```

1 début
2    $\tau_{sleep} \rightarrow \emptyset$ ;
3   pour  $P_i \in P$  faire
4      $\tau_{sleep}^i = \text{tache\_endormissement\_monoprocesseur}(C_{SleepMin})$ ;
5      $\tau_{sleep} \rightarrow \tau_{sleep} \cup \tau_{sleep}^i$ 
6   fin
7   pour  $P_i \in P$  faire
8      $\Gamma_{sleep}^i \rightarrow \Gamma_{sleep}^i \cup \text{Min}_{i=1..m}(\tau_{sleep}^i)$ 
9   fin
10 fin
  
```

---

Nous illustrons notre algorithme avec un exemple d'application. Le tableau ?? et les figures ?? et ?? representent un ensemble de tâches  $\Gamma$  ordonnançable par Deadline Monotonic dans un multiprocesseur à 2 processeur en partitionnement FirstFit où on a inséré deux taches d'endormissements  $\tau_{sleep}$  en locale et en globale.

$\tau_i$	$C_i$	$D_i$	$T_i$
1	5	10	10
2	7	15	21
3	2	22	24

TABLE 1.2 – Ensemble de tâche périodique

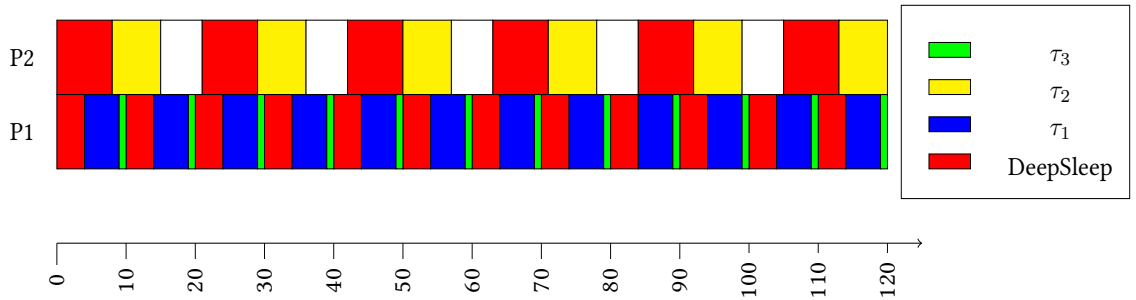


FIGURE 1.2 – Insertion locale de tâches d'endormissements dans un multiprocesseur

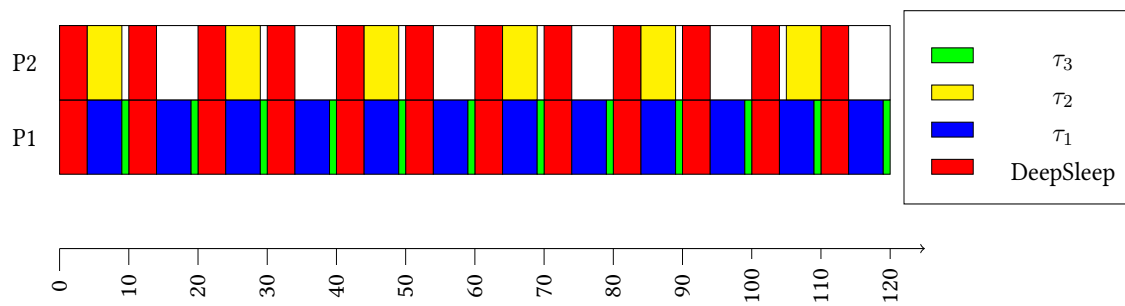


FIGURE 1.3 – Insertion globale de tâches d'endormissements dans un multiprocesseur

## Chapitre 2

# Endormissement de tâche sous priorité dynamique

### Sommaire

2.1	Introduction . . . . .	21
2.2	Les modèles de consommation d'énergie DVFS et DPM . . . . .	21
2.2.1	Consommation dynamique . . . . .	22
2.2.2	Consommation statique . . . . .	22
2.3	Les états C-states du processeur . . . . .	23
2.4	L'endormissement de processeur (Online VS Offline) . . . . .	24
2.5	Le Modèle d'endormissement de Dsouza . . . . .	24
2.5.1	Période d'harmonisation . . . . .	25
2.5.2	Algorithme d'ordonnancement « Energy-Saving Rate-Harmonized Scheduling » . . . . .	25
2.5.3	Energy-Saving Rate-Harmonized Scheduling . . . . .	25
2.5.4	Energy-Saving Rate-Harmonized Scheduling+ . . . . .	26
2.6	Conclusion . . . . .	27

### 2.1 modèle de tâches

Le modèle utilisé ici est le modèle de tâche sporadique.



Soit  $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$  un ensemble de tache, chaque tache  $\tau_i$  est caracterisé par  $\tau_i = (C_i, D_i, T_i)$  et :

- $\tau_i$  est sporadique.
- $C_i$  est le pire temps d'exécution de la tache<sub>*i*</sub>
- $D_i$  est l'échéance relative de la tache<sub>*i*</sub>
- $T_i$  est la periode de la tache<sub>*i*</sub>
- l'ensemble de tâches  $\Gamma$  est ordonnançable avec l'algorithme EarliestDeadlineFirst

## 2.2 Limitation de nombre de preemption

---

### Algorithme 4 : Insertion Taches Endormissement Dans Un Mono-processeur

---

**Données :** TaskSet de tâches sporadic  $\Gamma$

**Résultat :**  $Q(t)$

```

1 début
2   Posons  $\{t_1, t_2, \dots, t_n\}$  ensemble ordonné de  $\{D_i + lT_i, \forall l \in \mathbb{N}, 1 \leq i \leq n\}$  où
    $(t_k < t_{k+1}, \forall k)$   $Q(t_1) \leftarrow t_1 - \sum_{\tau_i \in \Gamma} DBF(\tau_i, t_1)$ ;
3   pour  $k \leftarrow 2, 3 \dots$  faire
4      $Q(t_k) \leftarrow \text{Min}(Q(t_{k-1}), t_k - \sum_{\tau_i \in \Gamma} DBF(\tau_i, t_k))$ ;
5     si  $Q(t_k) < 0$  alors
6       return faux;
7     fin
8   fin
9   return vrai;
10 fin
```

---

## 2.3 Le cas monoprocesseur

$\tau_i$	$C_i$	$D_i$	$T_i$
1	2	25	25
2	2	10	10

TABLE 2.1 – Ensemble de taches periodiques

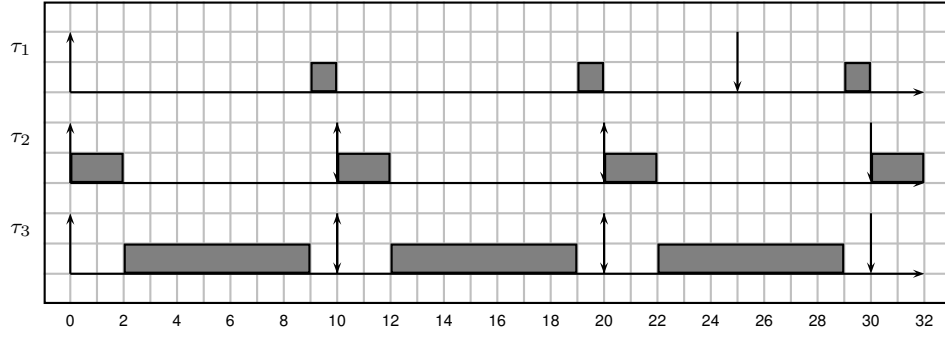


FIGURE 2.1 – Insertion de tâche dans un monoprocesseur

## 2.4 Le cas multiprocesseur

---

**Algorithme 5 :** Insertion Globale de Taches Endormissement Dans Une architecture Multiprocesseur

---

**Données :** Ensemble de processeur  $P$ , Temps Minimum d'exécution  $C_{SleepMin}$

**Résultat :** Ensemble de Tache d'endormissement  $\Gamma_{Sleep}$

```

1  début
2    pour  $P_i \in P$  faire
3       $\tau_{sleep}^i = \text{tache\_endormissement\_monoprocesseur}(C_{SleepMin})$ ;
4       $\Gamma_{Sleep}^i = \Gamma_{Sleep}^i \cup \tau_{sleep}^i$ 
5    fin
6  fin

```

---



---

**Algorithme 6 :** Insertion Globale de Taches Endormissement Dans Une architecture Multiprocesseur

---

**Données :** Ensemble de processeur  $P$ , Temps Minimum d'exécution  $C_{SleepMin}$

**Résultat :** Ensemble de Tache d'endormissement  $\Gamma_{Sleep}$

```

1  début
2     $\tau_{sleep} \rightarrow \emptyset$ ;
3    pour  $P_i \in P$  faire
4       $\tau_{sleep}^i = \text{tache\_endormissement\_monoprocesseur}(C_{SleepMin})$ ;
5       $\tau_{sleep} \rightarrow \tau_{sleep} \cup \tau_{sleep}^i$ 
6    fin
7    pour  $P_i \in P$  faire
8       $\Gamma_{sleep}^i \rightarrow \Gamma_{sleep}^i \cup \text{Min}_{i=1..m}(\tau_{sleep}^i)$ 
9    fin
10 fin

```

---

$\tau_i$	$C_i$	$D_i$	$T_i$
1	2	10	10
2	2	15	21

TABLE 2.2 – Insertion locale de tâches d'endormissements dans un multiprocesseur

## 2.5 insertion locale

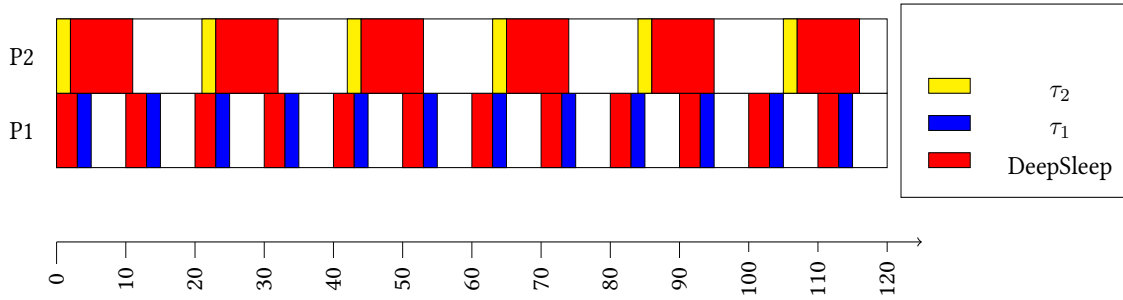


FIGURE 2.2 – Insertion globale de tâches d'endormissements dans un multiprocesseur

## 2.6 insertion globale

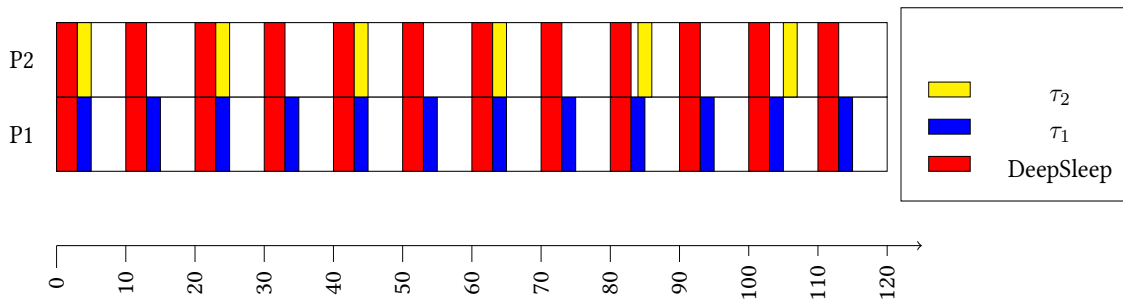


FIGURE 2.3

## Chapitre 3

# Expérimentations

### 3.1 La génération des tâches

#### 3.1.1 Cas monoprocesseur

##### Algorithmes UUnifast

L'algorithme UUnifast [22] est un algorithme mis au point pour la génération de taux d'utilisation sur monoprocesseur. Il génère une distribution uniforme de  $n$  taux d'utilisation non biaisés à partir du nombre de tâches  $n$  de l'ensemble et du taux d'utilisation processeur total souhaité  $U$ . UUnifast est un algorithme efficace de complexité  $O(n)$ . Nous rappelons qu'un ensemble au taux d'utilisation supérieur à 1 est trivialement non ordonnançable puisque l'utilisation processeur dépasse alors le temps maximal disponible.

##### Génération des périodes

Lors de la génération de tâches, le choix des périodes est un élément sensible pour les tests d'ordonnabilité. En effet, certains de ces tests basent leur analyse sur un intervalle de faisabilité. La longueur de cet intervalle dépend du plus petit commun multiple des périodes (ppcm) appelée l'hyperpériode. Si les périodes sont grandes, premières entre elles, l'hyperpériode explose. Le défi consiste donc à générer des périodes aléatoirement tout en limitant la taille de l'hyperpériode et c'est l'objet de la méthode de Goossens et Macq [?].

### Génération des échéances

Similairement à Goossens et Macq dans [?], nous generons les échéances des taches generées. Nous déterminons aléatoirement l'échéance dans un intervalle  $[0.75 \times T_i, T_i]$ . En résumé,  $Di = \{T_i \times C_i\} \times \text{aleatoire}(0.75 \times T_i, T_i)\} + C_i$  avec  $\text{aleatoire}(\text{dmin}, \text{dmax})$  retourne un nombre réel pseudo-aléatoire uniformément distribué sur l'intervalle  $[\text{dmin}, \text{dmax}]$  et où la fonction  $\text{arrondi}(x)$  retourne l'entier le plus proche de  $x$ .

### 3.1.2 Cas multiprocesseur

#### Algorithme UUnifast-Discard

La méthode UUnifast présentée dans le cas monoprocesseur n'est pas utilisée en contexte multiprocesseur, lorsque le taux d'utilisation du processeur  $U$  peut dépasser 1. En effet, lorsque que le taux d'utilisation total dépasse 1, UUnifast présente le risque de générer des taux d'utilisation par tâche supérieurs à 1. Tâche qu'il n'est alors possible d'ordonnancer sur aucun processeur. Pour y remédier, Davis et Burns [?] ont proposé une extension appelée UUnifast-Discard. Elle consiste simplement à employer UUnifast avec  $U$  supérieur à 1 et à rejeter les ensembles pour lesquelles au moins un taux d'utilisation par tâche est supérieur à 1. Son implantation est simple mais cette méthode a l'inconvénient d'être particulièrement inefficace lorsque  $U$  approche  $\frac{n}{2}$  [?].

## 3.2 La simulation

## 3.3 Discussions