Première partie

Contributions et Implementation

Chapitre 1

Endormissement de tache sous priorité Fixe

Sommaire

1.1 modéle de tâches

Le modèle utilisé ici est le modèle de tâche périodique de Liu et Layland défini au chapitre 1.
Soit $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ un ensemble de tache, chaque tache τ_i est caracterisé par $\tau_i = (C_i, D_i, T_i)$ et :

- τ_i est periodique.
- C_i est le pire temps d'execution de la tache$_i$.
- D_i est l'écheance relative de la tache$_i$.
- T_i est la periode relative de la tache$_i$.
- l'ensemble de tâches Γ est ordonnançable avec l'algorithme Deadline Monotonic.

1.2 Le cas monoprocesseur

Dans cette section nous allons inserer une tâche d'endormissement $\tau_{sleep} = \{C_{sleep}, D_{sleep}, T_{sleep}\}$ avec $C_{sleepMin} \leq C_{sleep} \geq C_{sleepMax}$ avec $C_{sleepMax} = (1-U) \times T_H$ et $\tau_{sleep} \cup \Gamma$ est ordonnançable avec Deadline Monotonic.
Pour cela nous presentons l'algorithme d'insertion de tache d'endormissement τ_{sleep} dans un taskset Γ.

Algorithme 1 : Insertion Taches Endormissement Dans Un Mono-processeur

Données : TaskSet Γ, Temps Minimum d'execution $C_{SleepMin}$, Temps d'execution Maximum $C_{SleepMax}$, Pas de decrementation ΔC
Résultat : Tache τ_{Sleep}

```
début
    C_Sleep ⟵ C_SleepMax ;
    D_Sleep ⟵ T_H ;
    T_sleep ⟵ T_H ;
    tant que Γ ∪ tache(C_Sleep, D_Sleep, T_H) est non ordonnançable et C_Sleep ≥ C_SleepMin
      faire
        C_Sleep ⟵ C_Sleep − ΔC ;
    fin
    τ_Sleep ⟵ CreerTache(C_Sleep, D_Sleep, T_sleep)
fin
```

Nous illustrons notre algorithme avec un exemple d'application. Le tableau 1.1 et la figure **??** represente un ensemble de tâches Γ ordonnançable par Deadline Monotonic où on a inseré une tache d'endormissement τ_{sleep}

τ_i	C_i	D_i	T_i
1	1	10	10
2	2	15	15

TABLE 1.1 – Ensemble de taches periodiques

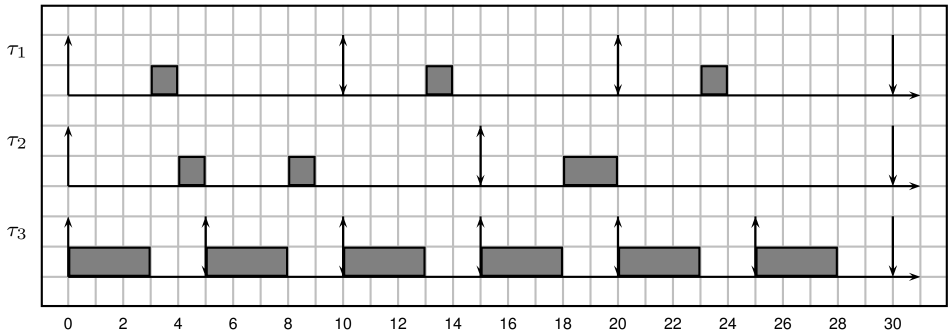

FIGURE 1.1 – Insertion de tache dans un monoprocesseur

1.3 Le cas multiprocesseur

Dans cette section nous allons inserer un ensemble de tâches d'endormissement $\{\tau^1_{sleep}, \tau^2_{sleep}, ..., \tau^m_{sleep}\}$ tel que $\tau^i_{sleep} = \{C^i_{sleep}, D^i_{sleep}, T^i_{sleep}\}$ dans m processeurs $P = \{P_1, P_2, ..., P_m\}$. Pour cela nous presentons nous presentons deux strategie d'insertion :

Insertion Locale : Chaque processeur$_i$ à sa propre tache d'endormissement τ^i_{sleep}

Insertion Globale : Tous les processeur ont une même tache d'endormissement $\tau_{sleep} = \tau^1_{sleep} = \tau^2_{sleep} = ... = \tau^m_{sleep}$

Les deux algorithmes representent l'inseretion 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 $\Gamma Sleep$

début
  pour $P_i \in P$ **faire**
    τ^i_{sleep} = tache_endormissement_monoprocesseur($C_{SleepMin}$) ;
    $\Gamma Sleep^i = \Gamma Sleep^i \cup \tau^i_{sleep}$
  fin
fin

Algorithme 3 : 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 $\Gamma Sleep$

début
  $\tau_{sleep} \rightarrow \emptyset$;
  pour $P_i \in P$ **faire**
    τ^i_{sleep} = tache_endormissement_monoprocesseur($C_{SleepMin}$) ;
    $\tau_{sleep} \rightarrow \tau_{sleep} \cup \tau^i_{sleep}$
  fin
  pour $P_i \in P$ **faire**
    $\Gamma sleep^i \rightarrow \Gamma sleep^i \cup Min_{i=1..m}(\tau^i_{sleep})$
  fin
fin

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

τ_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

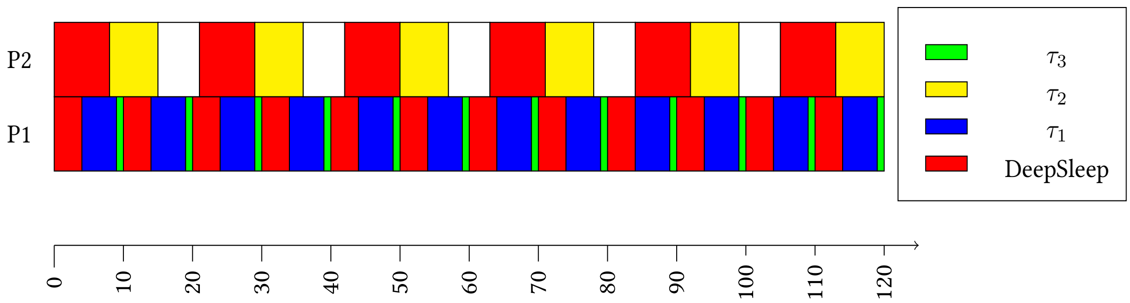

FIGURE 1.2 – Insertion locale de tâches d'endormissements dans un multiprocesseur

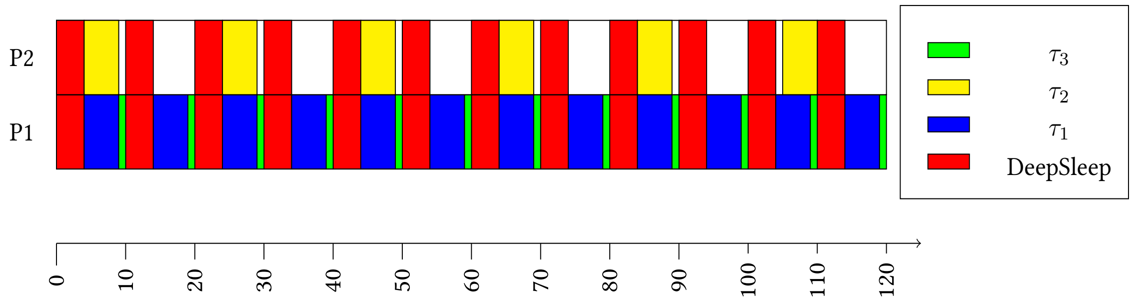

FIGURE 1.3 – Insertion globale de tâches d'endormissements dans un multiprocesseur

Chapitre 2

Endormissement de tache sous priorité dynamique

Sommaire

2.1 modéle de tâches

Le modèle utilisé ici est le modèle de tâche sporadique.

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

- τ_i est sporadique.
- C_i est le pire temps d'execution de la tache$_i$
- D_i est l'écheance relative de la tache$_i$
- T_i est la periode de la tache$_i$
- l'ensemble de tâches Γ est ordonnançable avec l'algorithme EarliestDeadlineFirst

2.2 Limitation de nombre de preemption

2.3 Le cas monoprocesseur

Algorithme 4 : Insertion Taches Endormissement Dans Un Mono-processeur

Données : TaskSet Γ, Temps Minimum d'execution $C_{SleepMin}$, Temps d'execution Maximum $C_{SleepMax}$, Pas de decrementation ΔC
Résultat : Tache τ_{Sleep}

début
  $C_{Sleep} \longleftarrow C_{SleepMax}$;
  $D_{Sleep} \longleftarrow T_H$;
  $T_{sleep} \longleftarrow T_H$;
  tant que $\Gamma \cup tache(C_{Sleep}, D_{Sleep}, T_H)$ est non ordonnançable et $C_{Sleep} \geq C_{SleepMin}$ **faire**
    $C_{Sleep} \longleftarrow C_{Sleep} - \Delta C$;
  fin
  $\tau_{Sleep} \longleftarrow CreerTache(C_{Sleep}, D_{Sleep}, T_{sleep})$
fin

τ_i	C_i	D_i	T_i
1	2	25	25
2	2	10	10

TABLE 2.1 – Ensemble de taches periodiques

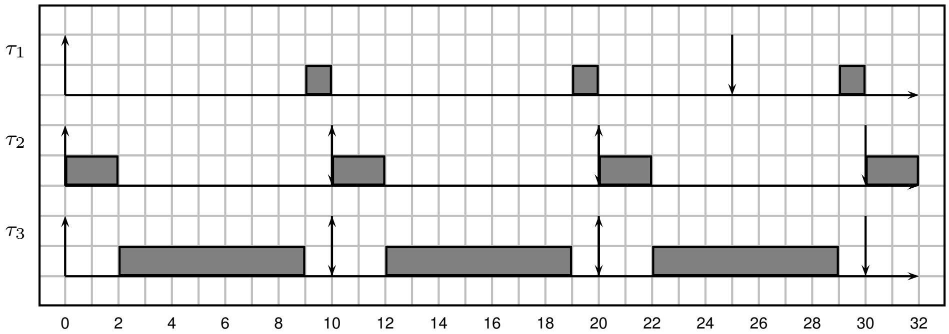

FIGURE 2.1 – Insertion de tache dans un monoprocesseur

2.4 Le cas multiprocesseur

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

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

```
début
    pour P_i ∈ P faire
        τ^i_sleep = tache_endormissement_monoprocesseur(C_SleepMin);
        ΓSleep^i = ΓSleep^i ∪ τ^i_sleep
    fin
fin
```

Algorithme 6 : 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 $\Gamma Sleep$

```
début
    τ_sleep → ∅;
    pour P_i ∈ P faire
        τ^i_sleep = tache_endormissement_monoprocesseur(C_SleepMin);
        τ_sleep → τ_sleep ∪ τ^i_sleep
    fin
    pour P_i ∈ P faire
        Γsleep^i → Γsleep^i ∪ Min_{i=1..m}(τ^i_sleep)
    fin
fin
```

τ_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

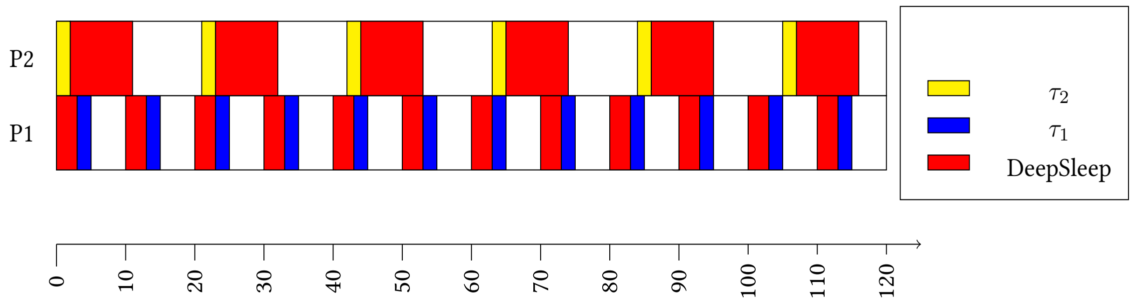

FIGURE 2.2 – Insertion globale de tâches d'endormissements dans un multiprocesseur

2.6 insertion globale

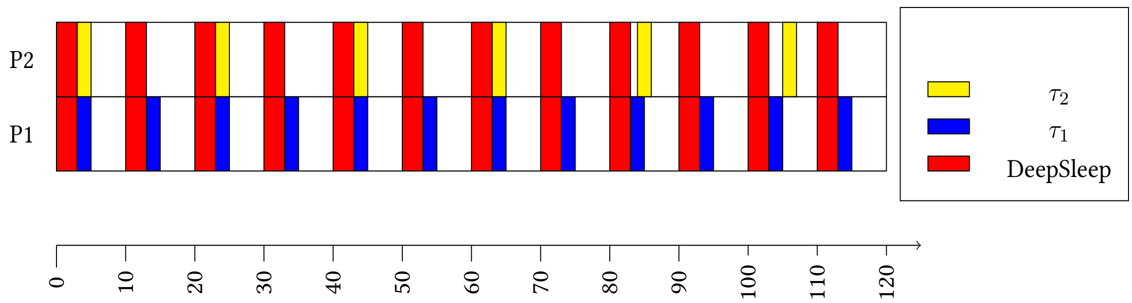

FIGURE 2.3

Chapitre 3

Expérimentations

3.1 La génération des taches

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'ordonnançabilité. 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'hyper-période 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, $Di = \{T_i \times C_i) \times aleatoire(0.75 \times T_i, T_i)\} + C_i$ avec aleatoire(dmin, dmax) retourne un nombre réel pseudo-aléatoire uniformément distribué sur l'intervalle [dmin, dmax] et où la fonction 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