Exercices de Systèmes Temps Réel

Giuseppe Lipari

October 9, 2015

Contents

1 Analyse d'ordonnançement temps réel

1.1 Fixed priority et Hyperplanes

Considérez le système temps réel suivante (ordonnanceur: *Fixed Priority* avec *Deadline Monotonic*)

Task	C	T	D
τ_1	1	5	5
τ_2	2	8	8
τ_3	4	18	12
τ_4	4	30	24

1. Calculez l'utilisation totale de l'ensemble de tache. Est-il supérieur à la borne U_{lub} ?
2. Calculez le temps de réponse de la tâche τ_3 et τ_4. Est-ce que le système est ordonnançable ?
3. Comment on peut réduire le temps de réponse de τ_3 ?
4. Utiliser la méthode des Hyperplanes pour établir l'ordonnaçabilité

5. Comment on peut augmenter le temps de calcul de la tache τ_3 en gardant l'ordonnançabilité de toutes les tâches?

6. En supposant que $C_4 = 8$, calculez combien on doit incrémenter la vitesse du processeur pour maintenir la tache τ_4 ordonnançable.

1.1.1 :Solution:

1. $U = 0.81$ et $U_{lub} = 0.76$, donc il est supérieure à la borne. En tout cas, on n'a pas le droit d'utiliser le test d'utilisation, parce que $D_3 < T_3$.

2. $R_3 = 8$, $R_4 = 15$. Le système est ordonnançable parce que tous les temps de réponse sont inférieurs aux échéances.

3. Il faut augmenter sa priorité. On peut augmenter sa priorité au maximum sans rater les autres échéances : donc le nouveau ordre devienne $\tau_3, \tau_1, \tau_2, \tau_4$. Les temps de réponse peut être réduit à $R_3 = C_3 = 4$; par contre, les temps de réponse des autres tâches changent: $R_1 = 5$, $R_2 = 8$. Le temps de réponse de τ_4 ne change pas (parce que on change pas sa position dans l'ordre de priorité).

4. On commence par la tâche τ_1. Il y a juste un contrainte:

$$C_1 \leq D_1$$

Pour la tâche τ_2, il faut considérer les points $\mathcal{P}_2 = \{5, 8\}$, donc 2 inégalités :

$$\begin{aligned} C_1 + C_2 &\leq 5 \\ 2C_1 + C_2 &\leq 8 \end{aligned}$$

Pour la tâche τ_3 il faut considérer le points $\mathcal{P}_3 = \{5, 8, 10, 12\}$:

$$\begin{aligned} C_1 + C_2 + C_3 &\leq 5 \\ 2C_1 + C_2 + C_3 &\leq 8 \\ 2C_1 + 2C_2 + C_3 &\leq 10 \\ 3C_1 + 2C_2 + C_3 &\leq 12 \end{aligned}$$

Finalement, pour la tâche τ_4, il faut considérer le points $\mathcal{P}_4 = \{5, 8, 10, 15, 16, 18, 20, 24\}$:

$$\begin{aligned} C_1 + C_2 + C_3 + C_4 &\leq 5 \\ 2C_1 + C_2 + C_3 + C_4 &\leq 8 \\ 2C_1 + 2C_2 + C_3 + C_4 &\leq 10 \\ 3C_1 + 2C_2 + C_3 + C_4 &\leq 15 \\ 4C_1 + 2C_2 + C_3 + C_4 &\leq 16 \\ 4C_1 + 3C_2 + C_3 + C_4 &\leq 18 \\ 4C_1 + 3C_2 + 2C_3 + C_4 &\leq 20 \\ 5C_1 + 3C_2 + 2C_3 + C_4 &\leq 24 \end{aligned}$$

Si on fait les substitutions avec les valeurs de temps de calculs, au moins un inégalité est respecté pour chaque tâche, donc le système est ordonnançable.

5. On fait la substitution des toutes les valeurs, sauf C_3, on obtient:

   Pour les tâches τ_1 et τ_2, les inégalités sont toujours vérifiés Pour la tâche τ_3, on obtient:

$$C_3 \leq 2$$
$$C_3 \leq 4$$
$$C_3 \leq 4$$
$$C_3 \leq 5$$

   On garde le maximum, parce que au moins un inégalité doit être respecté, donc $C_3 \leq 5$.

   Pour la tâche τ_4, on obtient:

$$C_3 \leq -2$$
$$C_3 \leq 0$$
$$C_3 \leq 0$$
$$C_3 \leq 4$$
$$C_3 \leq 4$$
$$C_3 \leq 4$$
$$C_3 \leq 3$$
$$C_3 \leq 4.5$$

   Encore une fois, on garde le maximum, $C_3 \leq 4.5$. Finalement, toutes les inégalités que on a gardés doivent être respectés, donc on prends le minimum. La solution finale est:

$$C_3 \leq 4.5$$

1.2 Blocage, PI

Les taches suivantes sont gérées par Fixed Priority (Deadline Monotonic) et avec Priority Inheritance.

Task	C	T	D	S_1	S_2	S_3	S_4	B
τ_1	1	5	5	1	-	-	-	
τ_2	2	10	10	-	1	1	-	
τ_3	5	30	15	-	-	3	1	
τ_4	7	40	30	2	3	1	-	
τ_5	10	100	60	-	1	-	2	

1. Calculez le temps de blocage de toutes les taches.
2. Vérifiez que le système n'est pas ordonnançable.

1.2.1 :Solution:

1. Voici le temps de blocage:

Task	C	T	D	S_1	S_2	S_3	S_4	B
τ_1	1	5	5	1	-	-	-	2
τ_2	2	10	10	-	1	1	-	6
τ_3	5	30	15	-	-	3	1	5
τ_4	7	40	30	2	3	1	-	2
τ_5	10	100	60	-	1	-	2	0

2. On essaye avec le temps de réponse :

$$R_1 = C_1 + B_1 = 3 \leq D_1$$

$$\begin{aligned} R_2^{(0)} &= C_1 + C_2 + B_2 = 9 \\ R_2^{(1)} &= 2C_1 + C_2 + B_2 = 10 \\ R_2^{(2)} &= 2C_1 + C_2 + B_2 = 10 \leq 10 \end{aligned}$$

$$\begin{aligned} R_3^{(0)} &= C_1 + C_2 + C_3 + B_3 = 13 \\ R_3^{(1)} &= 3C_1 + 2C_2 + C_3 + B_3 = 17 > 15 \end{aligned}$$

La tâche τ_3 peut rater son échéance, donc le système n'est pas ordonnançable.

1.3 Earliest Deadline First 1

Considérez le système suivant, avec ordonnancement EDF :

Task	C	D	T
τ_1	1	5	5
τ_2	2	9	9
τ_3	3	12	15
τ_4	4	10	18

1. Dites si le système est ordonnançable.
2. Réduisez au minimum le jitter de la tâche τ_3 en gardant l'ordonnançabilité du système.

1.3.1 :Solution:

1. Il faut vérifier avec le *demand bound function*.

$$L^* = \frac{1}{1-U} \max T_i - D_i = 43.43$$

On vérifie aussi le calcule du *idle time* :

$$W(0) = C_1 + C_2 + C_3 + C_4 = 10$$
$$W(1) = 2C_1 + 2C_2 + C_3 + C_4 = 13$$
$$W(2) = 3C_1 + 2C_2 + C_3 + C_4 = 14$$
$$W(3) = 3C_1 + 2C_2 + C_3 + C_4 = 14$$

Donc il faut s'arrêter à 14.

L	5	9	10	12
dbf	1	3	8	11

Le système est ordonnançable.

2. Pour réduire le jitter de τ_3 il faut réduire son échéance. Mais on ne sait pas de combien on peut la réduire. On essaie avec $D_3 = 6$. La borne ne dépend pas des échéances, donc il est toujours 14.

L	5	6	9	10
dbf	1	4	6	**11**

Donc, une tâche va rater son échéance. Observe qu'il n'est pas possible de avoir une échéance D_3 inférieur à 10. On essaie donc avec $D_3 = 11$.

L	5	9	10	11
dbf	1	3	8	11

Donc, $D_3 = 11$ est le minimum possible.

1.4 Earliest Deadline First 2

Considérez le système suivant, avec ordonnancement EDF :

Task	C	T	D
τ_1	1	5	5
τ_2	3	8	8
τ_3	3	15	10
τ_4	3	20	15

1. Verifiez l'ordonnançabilité du système
2. Calculez une borne supérieure au *jitter* pour la tâche τ_3
3. Essayez de réduire le *jitter* de la tâche τ_3 tout en gardant le système ordonnançable.

1.5 Earliest Deadline First + Blocage

Considérez le système temps réel suivant (ordonnancé par EDF) :

Task	C	D	T	R1	R2	R3	R4	Bloc. PI	Bloc. SRP
τ_1	1	6	6	0	1	0	1		
τ_2	2	10	10	2	0	1	0		
τ_3	3	16	16	0	0	2	2		
τ_4	4	18	24	2	1	1	0		

1. Dites si le système est ordonnançable sans considérer le blocage
2. Calculez le temps de blocage avec Priority Inheritance et avec Stack Resource Policy
3. Répétez l'analyse d'ordonnançabilité avec le temps de blocage avec le protocole SRP.

2 Programmation temps-réel

2.1 Code optionnel

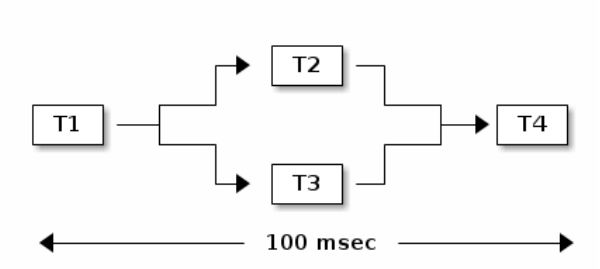
Un système temps-réel consiste de 3 tâches periodiques τ_1, τ_2, τ_3. Chaque job de chaque tache contient une première partie obligatoire et une deuxième partie optionnelle. Les trois tâches sont gérées par Fixed Priority : τ_1 a la priorité plus élevée, τ_2 la moyenne, τ_3 la moins élevée.

Normalement, chaque tâche exécute ses deux parties, obligatoire et optionnelle. Si la tâche τ_i rate son échéance, elle suspends l'exécution de la partie optionnelle pour les prochains N_i *jobs* (le paramètre N_i peut être configuré par le développeur). Si, après le N_i-ème *job*, il y a encore une échéance raté, le programme se termine avec un erreur ; par contre, si la dernière échéance est respecté, la tâche reprenne à exécuter la partie optionnelle à partir du prochain job.

1. Écrire le code de la i-ème tâche de manière générique
2. Expliquez les conditions où une telle méthode n'est pas suffisante à corriger le problème de la surcharge. Démontrez-le avec un exemple numérique :
   - choisissez les périodes est les temps de calculs des les 3 tâches
   - montrez que, après N_i *jobs*, il y a encore une échéance ratée.

2.2 Activation séquentielle

Un système temps réel est composée de 5 tâches. La tâche τ_1 est périodique de période égal à 100 milliseconds. Les tâches τ_2 et τ_3 sont activées par la tâche τ_1 peu avant la fin de chaque *job*. La tâche τ_4 doit être activée quand le deux tâches τ_2 et τ_3 ont terminées. La tâche doit se terminer entre 100 milliseconds de l'activation de τ_1. Le schéma est montrée dans la figure suivante :



Finalement, la tache τ_5 est périodique avec une période égal à 60 milliseconds.

1. Écrivez le code pour les tâches $\tau_1, \tau_2, \tau_3, \tau_4$. La tâche τ_4 vérifie si l'échéance est ratée.
2. Prenez les temps d'exécution dans le tableau suivant :

Task	C	D	T	priorité
τ_1	10	–	100	
τ_2	10	–	–	
τ_3	10	–	–	
τ_4	20	100	–	
τ_5	5	60	60	

Donnez les priorité à les tâches pour garantir l'ordonnançabilité du système.

2.3 Changement de mode

Un système temps réel est composé par 2 modes de fonctionnement. Les tâches est les modes sont montrez dans le tableau suivant:

Task	C	T	Mode
τ_1	1	5	A, B
τ_2	2	8	A
τ_3	3	9	B
τ_4	3	12	A
τ_5	2	10	B
τ_6	3	21	A,B

En outre, le système contient une tâche de gestion, avec un temps d'exécution négligeable qui s'occupe de changer le mode du système si nécessaire.

L'algorithme d'ordonnancement est Fixed Priority, et le système se trouve initialement dans le mode A.

Quand un certain évènement externe se présents, la tâche de gestion de mode fait la transition vers le mode B.

1. Écrivez le pseudo-code pour les taches $\tau_2, \ldots, \tau_5$ (il faut écrire un modèle de tâche).
2. Écrivez le pseudo-code pour la tâches de gestion des modes.
3. Calculez le maximum temps de transition (c'est à dire, du moment où l'évènement se déclenche, au moment quand toutes le tâches dans le mode A ont étés désactivés et le tâches dans le mode B ont été activées) en utilisant le protocole "idle-time".

2.4 Broadcast communication

Dans un système temps réel, une tâche périodique τ_1 mémorises une donnée scalaire dans un structure de donnée `struct data` en utilisant la fonction :

```
void insert(struct data *s, double d);
```

Deux taches, τ_2 et τ_3, lisent les données en utilisant la fonction :

```
double get(struct data *s, int index)
```

Si le paramètre `index` vaut 0, l'invocation est fait par la tâche τ_2 ; s'il vaut 1, l'invocation est fait par la tâche τ_3.

Une donnée a été *consommée* quand toutes les deux tâches ont bien lit la donnée. La structure de donnée peut mémorisé que un maximum de N données consécutives qui ont été produites mais pas encore consommées. Dans le cas où la structure est plein, la tâche τ_1 doit se bloquer dans l'appel `insert()`.

1. Écrivez le code de la structure de données, et des fonctionnes `insert()` et `get()` en utilisant le mécanisme des *monitor* avec `pthread_mutex_t` et `pthread_cond_t`.
2. Écrivez la structure du corps de la tâche τ_1 qui relève la condition de *deadline miss*.

2.5 Preemption

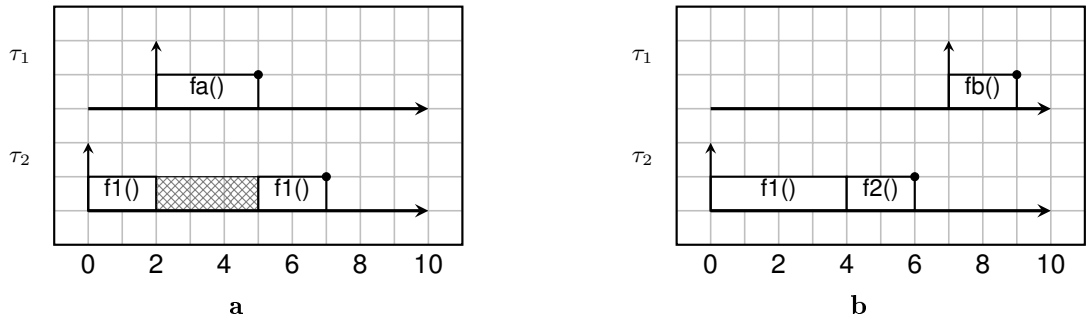


Figure 1: Deux cas: a) la tâche τ_1 a fait preemption sur τ_2; b) la tâche τ_1 n'a pas fait preemption sur τ_2

Un système temps réel est composé par deux tâches périodiques : la tâche τ_1 avec priorité élevé et période égale à 10 milliseconds, et la tâche τ_2 à priorité moins élevé et avec période égale à 25 milliseconds.

Normalement la tâche τ_2 exécutes la fonction `f1()`

- si pendant l'exécution de cette fonction la tâche τ_1 fait preemption, alors la tâche τ_1 se suspends quand `f1()` se termine (voir la figure 1.a);
- s'il n'y a pas de preemption pendant l'exécution de `f1()`, il continue avec l'exécution de la fonction `f2()` avant de se suspendre (voir la figure 1.b).

La tâche τ_1 contrôle au début s'il a interrompu τ_2 :

- si oui, il exécutes la fonction `fa()` (figure 1.a) ;
- si non, il exécutes la fonction `fb()` (figure 1.b).

1. Écrivez le code de les deux tâches.