# Systèmes et programmation temps-réels

Andréas Guillot

March 31, 2017

## Informations

Chaque exercice a son propre makefile.

Lancement des programmes avec sudo taskset -c 0 ./program

Selon *man sched_get_priority_min* : *"Processes with numerically higher priority values are scheduled before processes with numerically lower priority values"*.

## Exercice 1 :

La figure 1 montre un exemple d'exécution du premier exercice.

On peut observer que la précision est bien plus faible que la nanoseconde annoncée par *clock_getres()*, mais qu'elle est plutôt de l'ordre de la milliseconde.

## Exercice 2 :

Dans cette partie chaque question aura son sous-dossier. Le code qui a changé est délimité par les balises *// ADDED*.

### Question 1 :

La figure 2 montre un exemple d'exécution. On peut aussi observer une préemption se produire lorsque T2 démarre et que T1 prends la main pendant toute la durée de son exécution.

### Question 2 :

la figure 3 montre un exemple d'exécution. Le nombre d'échéances ratées dépends largement du calibrage et du taux d'utilisation du processeur. L'exemple montre comment ces contraintes font que même T1 peut rater des échéances, alors que T2 devrait normalement être le seul à en faire.

```
andreas@pc:~/Documents/signals/tp3/exo1$ sudo taskset -c 0 ./program
Clock resolution: 0s, 1ns
Temps écoulé = 1s, 1145192404 ns
Temps écoulé = 2s, 2145237972 ns
Temps écoulé = 3s, 3145190612 ns
Temps écoulé = 4s, 4145193172 ns
Temps écoulé = 5s, 5145193940 ns
Temps écoulé = 6s, 6145201876 ns
Temps écoulé = 7s, 7145193172 ns
Temps écoulé = 8s, 8145207764 ns
Temps écoulé = 9s, 9145193940 ns
Temps écoulé = 10s, 10145193684 ns
Temps écoulé = 11s, 11145193428 ns
Temps écoulé = 12s, 12145201620 ns
Temps écoulé = 13s, 13145194452 ns
Temps écoulé = 14s, 14145178324 ns
Temps écoulé = 15s, 15145182164 ns
Temps écoulé = 16s, 16145236436 ns
Temps écoulé = 17s, 17145194452 ns
Temps écoulé = 18s, 18145193428 ns
Temps écoulé = 19s, 19145178580 ns
Temps écoulé = 20s, 20145195988 ns
Temps écoulé = 21s, 21145182420 ns
Temps écoulé = 22s, 22145223124 ns
Temps écoulé = 23s, 23145180884 ns
Temps écoulé = 24s, 24145188052 ns
Temps écoulé = 25s, 25145176788 ns
Temps écoulé = 26s, 26145221588 ns
Temps écoulé = 27s, 27145193940 ns
Temps écoulé = 28s, 28145196244 ns
Temps écoulé = 29s, 29145195220 ns
```

Figure 1: Exemple d'exécution de l'exercice 1

```
andreas@pc:~/Documents/signals/tp3/exo2/q1$ sudo taskset -c 0 ./program
Calibrage k = 11159

Démarrage T1 a 1490973054 s et 338638918 ns.
Fin de T1 a 1490973054 s et 340407183 ns
Durée : 1768265ns
        Démarrage T2 a 1490973054 s et 340464419 ns.
Démarrage T1 a 1490973054 s et 342590194 ns.
Fin de T1 a 1490973054 s et 344361269 ns
Durée : 1771075ns
        Fin de T2 a 1490973054 s et 345500971 ns
        Durée : 5036552ns
```

Figure 2: Exemple d'exécution de la première question

```
andreas@pc:~/Documents/signals/tp3/exo2/q2$ sudo taskset -c 0 ./program | grep rat
T1 -----> 1 échéances ratées
T2 -----> 10 échéances ratées
T2 -----> 4 échéances ratées
T2 -----> 2 échéances ratées
T2 -----> 1 échéances ratées
T2 -----> 1 échéances ratées
```

Figure 3: Exemple d'échéances ratées

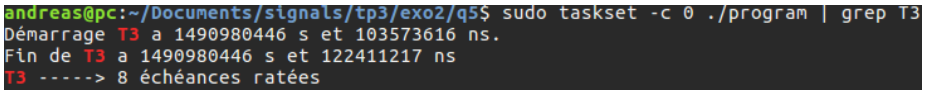


Figure 4: Exemple de T3 qui parvient tant bien que mal à s'exécuter dans un programme non ordonnançable

## Question 3 :

Les résultats sont bien meilleurs : cela s'explique par le fait que *printf()* soit très coûteux, et que les appels successifs à cette fonction va ralentir notre programme et ainsi fausser nos résultats.

## Question 4 :

La solution que j'ai choisie pour éliminer les appels à *printf()* consiste à créer un autre processus qui va se charger d'afficher les chaînes de caractères qu'il aura reçu d'une file de messages.

## Question 5 :

Ajout une tâche avec une capacité égale à 6 et une période égale à 8 de plus faible priorité ne sera pas ordonnançable.

En effet, on peut utiliser le test RMA pour voir que la valeur est tellement importante qu'on peut conclure que ce ne sera pas ordonnançable sans avoir besoin de faire le test RMA étendu :

$$\frac{2}{4} + \frac{4}{6} + \frac{6}{8} = 1.91$$

De plus, on voit que le processeur est constamment occupé par T1 et T2 dans les 8 premières secondes d'exécution du programme vu que T1 (de plus haute priorité) va être exécuté pendant 4 unités de temps et que T2 va occuper les 4 unités restantes.

Néanmoins, on peut voir T3 s'exécuter, comme dans la figure 4.