

# PTR Laboratoire 3

---

## Linux et Xenomai

---

Spinelli Isaia

le 09 Novembre 2019

### PTR Laboratoire 3

[Linux et Xenomai](#)

[Introduction et objectif](#)

[Étape 1 Priorité des threads et Linux](#)

[Étape 2 Priorités](#)

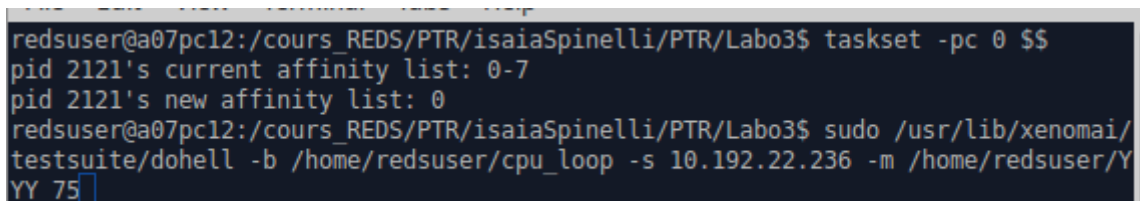
[Étape 3 Xenomai](#)

[Conclusion](#)

## Introduction et objectif

---

Lors du dernier labo nous avons testé les capacités d'une application à servir des tâches périodiques sous différentes conditions. Nous allons maintenant effectuer des tests et récolter des données dans un environnement plus chargé en termes d'utilisation du temps processeur. Afin de créer un tel environnement, nous allons utiliser les commandes suivantes :



```
redsuser@a07pc12:/cours_REDS/PTR/isaiaSpinelli/PTR/Labo3$ taskset -pc 0 $$
pid 2121's current affinity list: 0-7
pid 2121's new affinity list: 0
redsuser@a07pc12:/cours_REDS/PTR/isaiaSpinelli/PTR/Labo3$ sudo /usr/lib/xenomai/
testsuite/dohell -b /home/redsuser/cpu_loop -s 10.192.22.236 -m /home/redsuser/Y
YY 75
```

Les options du script dohell sont les suivantes :

1. -b hackbench : chemin de l'application ou du script à stresser
2. -s server [-p port] : destination à laquelle des paquets réseau seront envoyés via nc (ou netcat) ;  
le port par défaut est 9
3. -m folder : répertoire (ou point de montage) dans lequel un fichier de 100MB sera créé et rempli  
de 0 grâce à une boucle infinie
4. duration : temps d'exécution du script, en secondes

## Étape 1 Priorité des threads et Linux

---

Pour tous le laboratoire j'ai choisi une période de **500 us**. Donc, afin de l'exécuter longtemps (70 secondes) il faut faire 140'000 mesures.

Il est important de lancer le timer sur le même cœur que le programme "dohell" afin de le perturber au maximum. Voici la commande à exécuter :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ taskset -pc 0 $$
pid 2668's current affinity list: 0
pid 2668's new affinity list: 0
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ ./timer 140000 500 | ./summary
```

Voici-ci dessous 2 résultats différents:

- 1 : Timer sans perturbation
- 2 : Timer avec perturbation

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ ./timer 1000 500 | ./summary
----- summary1.c -----
Total of 1000 values
Minimum = 496293.000000 (position = 111)
Maximum = 502361.000000 (position = 70)
Sum = 499998302.000000
Mean = 499998.302000
Variance = 62535.536957
Std Dev = 250.071064
CoV = 0.000500
-----

redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ ./timer 1000 500 | ./summary
----- summary1.c -----
Total of 1000 values
Minimum = 171297.000000 (position = 157)
Maximum = 20208066.000000 (position = 60)
Sum = 940998315.000000
Mean = 940998.315000
Variance = 4689713432062.937500
Std Dev = 2165574.619371
CoV = 2.301359
-----
```

On peut clairement voir que sans perturbation le timer est correcte. Par contre, avec de la perturbation, la moyenne double donc il y a une erreur de 100%. De plus, la variance prend une valeur extrême. On ne peut clairement pas faire de temps réel avec ce genre de timer.

Afin d'avoir une mesure sur plus d'une minute avec perturbation, voici la commande et le résultat :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ ./timer 140000 500 | ./summary
----- summary1.c -----
Total of 140000 values
Minimum = 1794.000000 (position = 52019)
Maximum = 36604739.000000 (position = 28111)
Sum = 116507998301.000000
Mean = 832199.987864
Variance = 6345807020660.148438
Std Dev = 2519088.529739
CoV = 3.027023
-----
```

On peut voir que la moyenne est toujours complètement fausse et la variance toujours très élevée.

## Étape 2 Priorités

Maintenant que nous avons un environnement pour lancer les expériences, nous allons l'utiliser sur

différentes solutions. Dans cette étape, nous avons écrit un programme qui exécute une tâche périodique dans une tâche haute priorité et qui utilise la fonction `nanosleep()` pour attendre une période de temps à l'intérieur d'une boucle.

Remarque : La commande pour la perturbation est toujours la même décrite dans l'introduction.

Voici la commande à taper pour lancer les mesures :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ taskset -pc 0 $$
pid 13368's current affinity list: 0-7
pid 13368's new affinity list: 0
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ sudo ./Etape2 | ./summary
```

Remarque : la commande sudo est importante afin de pouvoir changer la priorité du thread.

Pour commencer, voici les résultats sans perturbation :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ sudo ./Etape2 | ./summary
----- summary1.c -----
Total of 140000 values
  Minimum = 501298.000000 (position = 120398)
  Maximum = 128144156.000000 (position = 47187)
  Sum      = 78461436036.000000
  Mean     = 560438.828829
  Variance = 235460459950.827454
  Std Dev  = 485242.681502
  CoV      = 0.865826
-----
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$
```

On peut voir que la variance est énorme et que la moyenne n'est pas précise. Donc de base on peut constater que le nanosleep() est moins précis qu'un timer ce qui semble correcte.

Ensuite, voici les résultats avec perturbations :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ taskset -pc 0 $$
pid 13368's current affinity list: 0-7
pid 13368's new affinity list: 0
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ sudo ./Etape2 | ./summary
----- summary1.c -----
^C
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo3/code$ sudo ./Etape2 | ./summary
----- summary1.c -----
Total of 140000 values
  Minimum = 501344.000000 (position = 86020)
  Maximum = 1600513.000000 (position = 7330)
  Sum      = 70405098950.000000
  Mean     = 502893.563929
  Variance = 322607034.489929
  Std Dev  = 17961.264835
  CoV      = 0.035716
-----
```

On peut voir que les résultats sont meilleurs. La moyenne est plus précise et la variance est moins extrême. Je ne saurais pas expliquer pourquoi mais il reste évident que pour du temps réel cette solution semble toujours pas bonne.

## Étape 3 Xenomai

Maintenant le but est de faire la même chose mais via l'interface native de Xenomai. Etant donné que celui-ci prend en charge le temps réel, on espère que les résultats seront bien plus précise que précédemment.

Remarque : La commande pour la perturbation est toujours la même décrite dans l'introduction.

Voici la commande à taper pour lancer les mesures :

```
redsuser@a07pc12:/cours_REDS/PTR/isaiaSpinelli/PTR/Labo3/code$ sudo ./xenomai_timer | ./summary
```

On peut voir ici les résultats avec perturbations :

```
redsuser@a07pc12:/cours_REDS/PTR/isaiaSpinelli/PTR/Labo3/code$ sudo ./xenomai_timer | ./summary
----- summary1.c -----
Total of 140000 values
Minimum = 483913.000000 (position = 61803)
Maximum = 517255.000000 (position = 61802)
Sum      = 69999976635.000000
Mean     = 499999.833107
Variance = 456986.549042
Std Dev  = 676.007803
CoV      = 0.001352
-----
```

La moyenne est extrêmement précise ! On voit que le temps réel est un autre monde que le non temps réel.

De plus, la variance reste relativement faible pour un temps de 500us.

Remarque:

Tous les codes sont en annexe et commentés.

Malheureusement je n'ai pas mis d'histogramme car je trouve que ceux fourni par octave n'était pertinent dû au fait de la qualité du graphe.

## Conclusion

---

Le timer reste un peu mieux qu'un simple appelle à une fonction nanosleep() mais cela dit, le timer reste très loin des performances que l'interface de Xenomai propose. On constate que le temps réel fait une grosse différence de précision.