

# PTR Laboratoire 2

---

Spinelli Isaia

le 16 Octobre 2019

## Gettimeofday et Gettimeofday2

---

3. On comparant les résultats de gettimeofday et gettimeofday2, On peut voir que le 2ème récupère des différences de temps plus faible. Ce qui est normal car il "perd" pas de temps à afficher entre chaque capture des résultats.
4. Etant donné que le fonction gettimeofday() manipule des structures avec des secondes et des microsecondes, on peut facilement imaginer que la précision tourne autour de la microsecondes (1us).

En annexe, le code dans "gettimeofday2.c"

## Horloges Posix

---

Après avoir modifié le fichier gettimeofday.c afin d'utiliser la fonction clock\_gettime() proposée par Posix, j'ai pu les comparer.

Donc, la comparaison entre la fonction proposée par Posix ("clock\_gettime") et gettimeofday() est claire, Posix propose une méthode avec une précision de **1ns** ce qui est 1000x meilleures que la précision de gettimeofday().

Finalement, la différence entre les horloges de Posix (CLOCK\_REALTIME, CLOCK\_MONOTONIC, CLOCK\_PROCESS\_CPUTIME\_ID, CLOCK\_THREAD\_CPUTIME\_ID) sont surtout à l'intervalle de quand la clock commence. Par exemple, la clock CLOCK\_THREAD\_CPUTIME\_ID, va commencer lors du démarrage du thread. Donc, nous affichera plutôt des secondes aux alentours de 0. Contrairement à la CLOCK\_REALTIME qui représente les secondes et les nanosecondes depuis l'époque.

Remarque : CLOCK\_REALTIME\_HR et CLOCK\_MONOTONIC\_HR ne compile plus.

## Développement : timer

---

sigaction (SIGALRM, &sa, NULL); =>

Premièrement, il va indiquer que lorsque le signal SIGALRM sera reçu, il faudra exécuter la fonction donnée (timer\_handler).

Ensuite, il initialise la structure timer de type struct itimerval qui permet de définir un temps de début et un temps d'intervalle.

setitimer (ITIMER\_REAL, &timer, NULL); =>

Finalement, il va mettre en place un "minuteur" en fonction de la structure initialisée plutôt. Ce minuteur enverra en fonction du type de la clock choisis un signal qui sera le même utilisé plus tôt (SIGALRM) qui déclenchera donc la fonction prédéfinie.

En bref, après 250ms, ce logiciel appelle la fonction souhaitée toutes les 250 ms.

En annexe, il y a le code avec quelques modifications. (timer\_Example.c)

## Modifications

Il nous est demandé d'écrire un programme qui prend en entrée le nombre de mesures à faire et un temps en microsecondes. De programmer un timer périodique CLOCK\_REALTIME qui affiche sur la sortie standard le temps écoulé entre deux différentes occurrences.

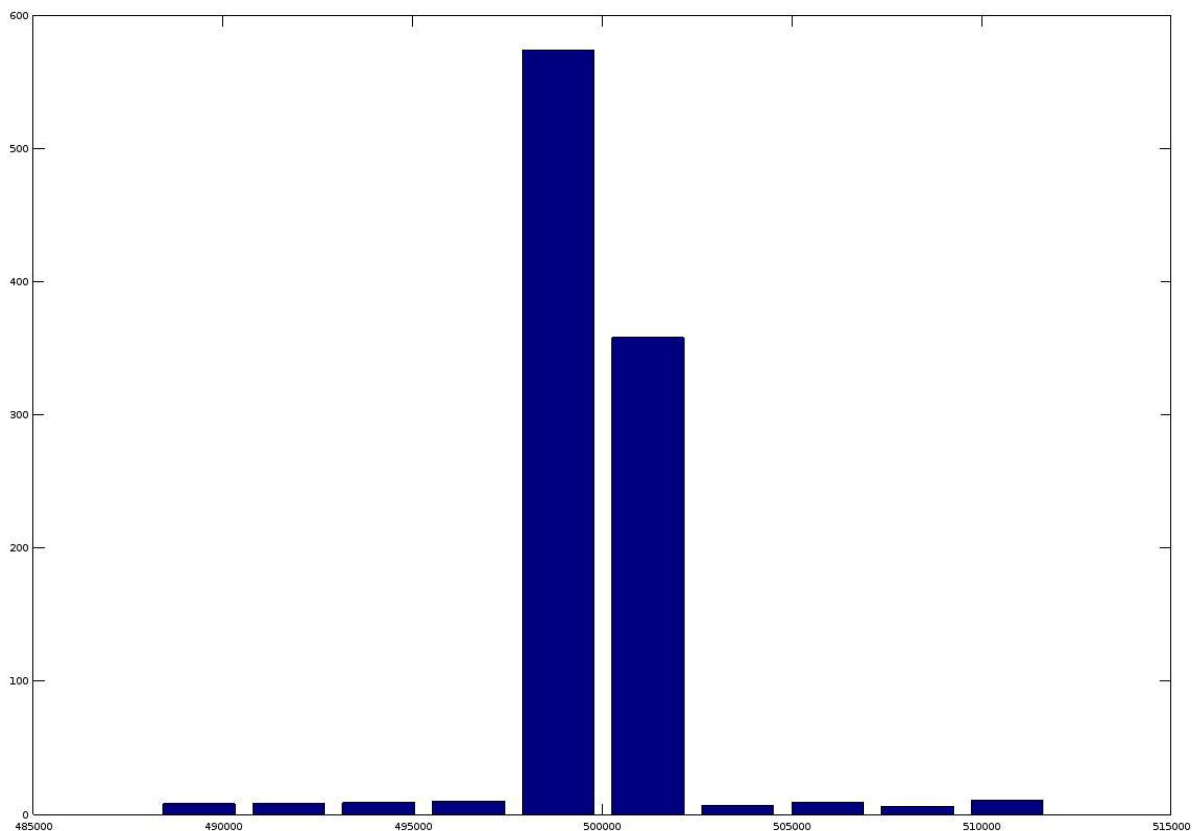
le code est en annexe. (timer.c)

Remarque : Pour que la programme compile il a fallu utiliser "-lrt" afin qu'il connaisse les fonctions "timer\_create" et "timer\_settime"

## Mesures

Résultat pour une intervalle de 500 us :

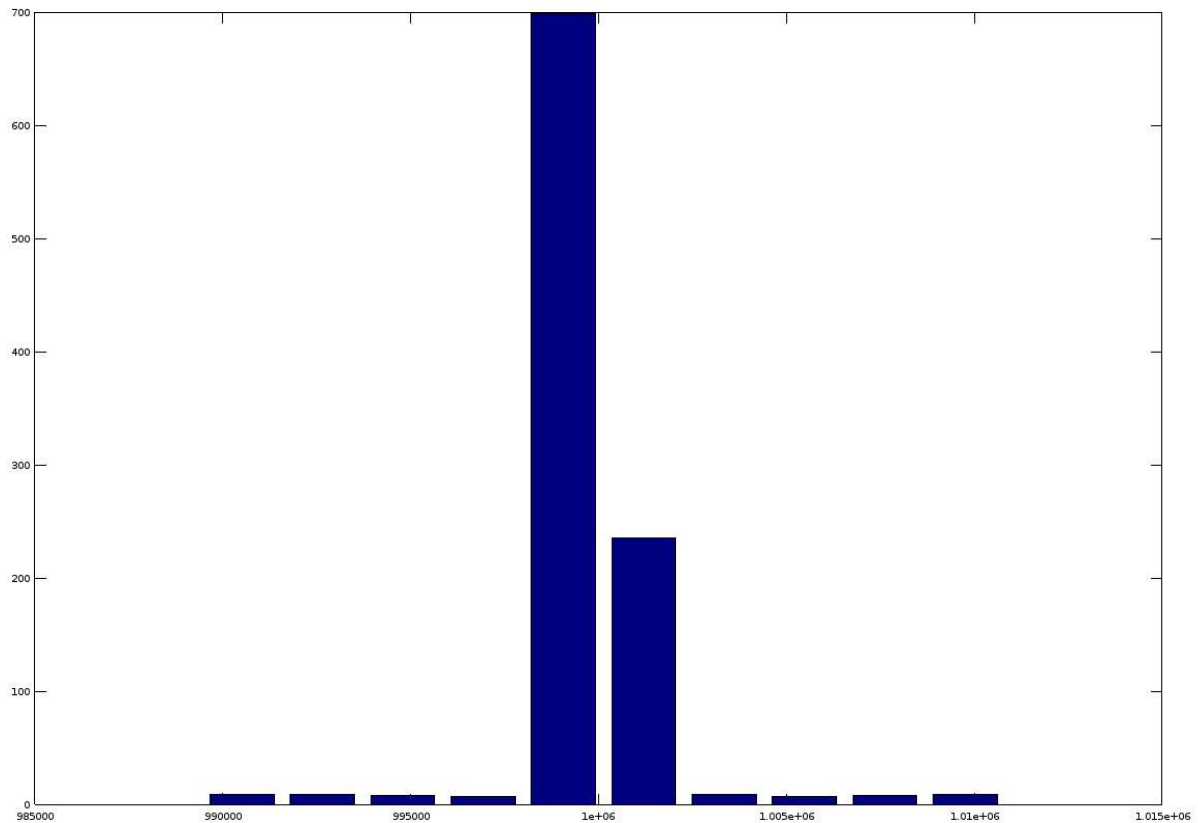
```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./timer 1000 500 > t500.dat
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./summary < t500.dat
----- summary1.c -----
Total of 1000 values
Minimum = 488183.000000 (position = 285)
Maximum = 511860.000000 (position = 284)
Sum      = 499998080.000000
Mean     = 499998.080000
Variance = 4145144.556152
Std Dev  = 2035.962808
CoV      = 0.004072
-----
```



Résultat pour une intervalle de 1000 us :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./summary < t1000.dat  
----- summary1.c -----
```

```
Total of 1000 values  
Minimum = 989444.000000 (position = 130)  
Maximum = 1010825.000000 (position = 129)  
Sum      = 999987910.000000  
Mean     = 999987.910000  
Variance = 3348309.811768  
Std Dev  = 1829.838739  
CoV      = 0.001830
```

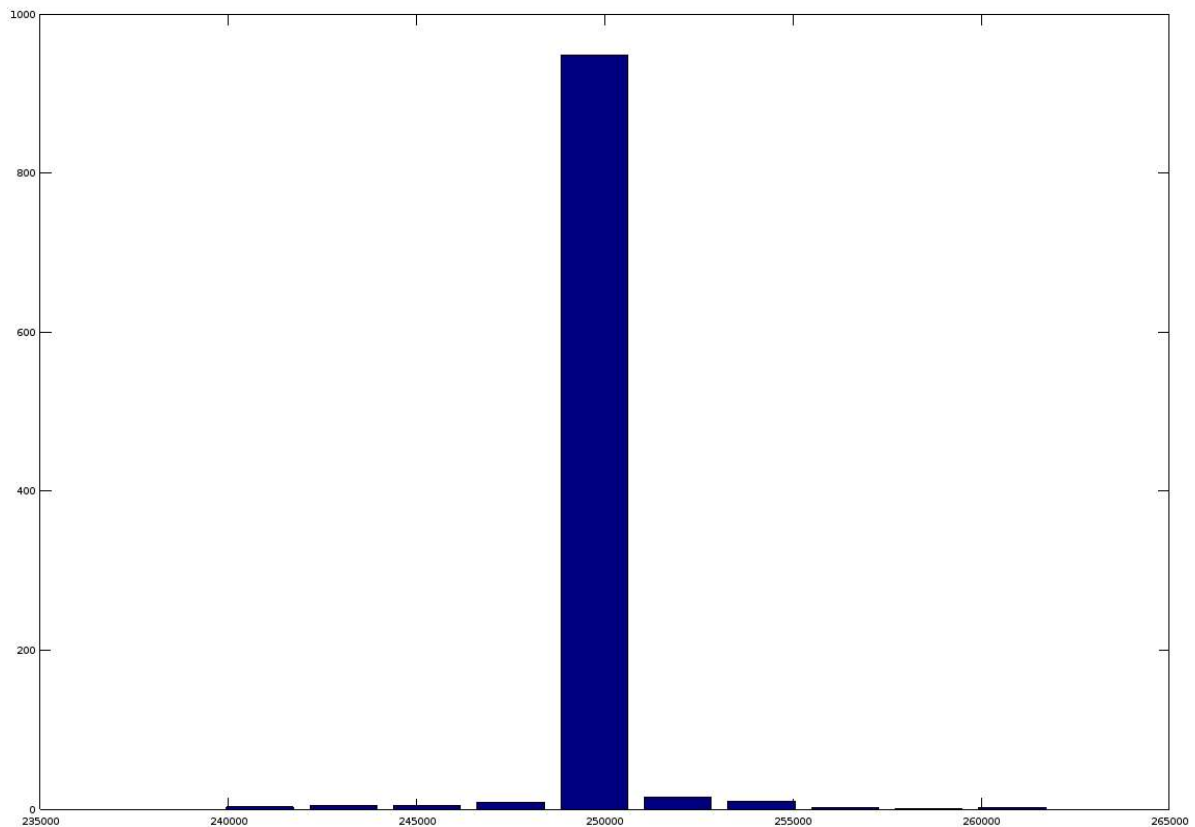


Résultat pour une intervalle de 250 us :

```

reduser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./timer 1000 1000 > t1000.dat
reduser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./summary < t250.dat
----- summary1.c -----
Total of 1000 values
Minimum = 239726.000000 (position = 443)
Maximum = 261937.000000 (position = 167)
Sum      = 249997848.000000
Mean     = 249997.848000
Variance = 1290158.755043
Std Dev  = 1135.851555
CoV      = 0.004543
-----

```



Remarque:

On peut facilement voir avec les histogrammes que les résultats sont correctes. Il y a quelques valeurs à cotés mais ceci est négligeable.

Test pour 1000 mesures avec une intervalle de 1ms 2 fois de suite:

```

redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo ./timer 1000 1000 | ./summary
----- summary1.c -----
$ Total of 1000 values
  Minimum = 991988.000000 (position = 29)
  Maximum = 1006734.000000 (position = 28)
  Sum      = 999998523.000000
  Mean     = 999998.523000
  Variance = 307311.268433
  Std Dev  = 554.356626
  CoV      = 0.000554
-----

redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 990159.000000 (position = 545)
  Maximum = 1009822.000000 (position = 975)
  Sum      = 999998659.000000
  Mean     = 999998.659000
  Variance = 462440.608032
  Std Dev  = 680.029858
  CoV      = 0.000680
-----

```

Remarque :

On peut constater que la moyenne est généralement assez précise. Par contre, la variance et le standard déviation n'est pas trop régulier. Il peut vite y avoir des pics (entre 200 - 1200 environ).

## Perturbations

---

Pour cette étape, j'ai surtout regardé le standard déviation afin de voir facilement les perturbations qui pourraient être causées.

### niceness

nice avec plusieurs coeurs :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo nice -n -18 ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 996515.000000 (position = 35)
  Maximum = 1003460.000000 (position = 34)
  Sum      = 999998744.000000
  Mean     = 999998.744000
  Variance = 110717.484253
  Std Dev  = 332.742369
  CoV      = 0.000333
-----

redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo nice -n 18 ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 997270.000000 (position = 50)
  Maximum = 1003922.000000 (position = 17)
  Sum      = 999997707.000000
  Mean     = 999997.707000
  Variance = 92542.347168
  Std Dev  = 304.207737
  CoV      = 0.000304
-----
```

nice à un cœur :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo nice -n 19 ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 989711.000000 (position = 983)
  Maximum = 1010342.000000 (position = 81)
  Sum      = 999998023.000000
  Mean     = 999998.023000
  Variance = 485476.590088
  Std Dev  = 696.761502
  CoV      = 0.000697
-----

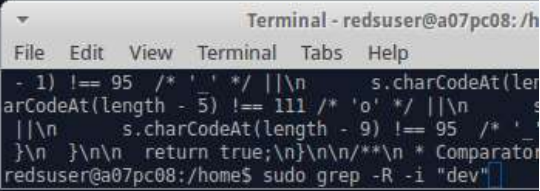
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo nice -n -18 ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 990339.000000 (position = 75)
  Maximum = 1010057.000000 (position = 74)
  Sum      = 999998644.000000
  Mean     = 999998.644000
  Variance = 407583.868164
  Std Dev  = 638.422954
  CoV      = 0.000638
-----
```

Nous pouvons voir les résultats une fois avec une niceness de -18 et une autre avec 18 ou 19. On peut remarquer qu'il n'y a pas de différence sur un cœur ou plusieurs.

nice + grep (1: avec grep - 2: sans grep) :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo nice -n 19 ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 987557.000000 (position = 506)
  Maximum = 1012589.000000 (position = 505)
  Sum      = 999998797.000000
  Mean     = 999998.797000
  Variance = 2537819.881592
  Std Dev  = 1593.053634
  CoV      = 0.001593
-----

redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo nice -n 19 ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 991017.000000 (position = 622)
  Maximum = 1008954.000000 (position = 621)
  Sum      = 999998410.000000
  Mean     = 999998.410000
  Variance = 584392.342163
  Std Dev  = 764.455585
  CoV      = 0.000764
-----
```





Par contre, en faisant une grosse opération, on peut voir la différence. La variance est plus élevée (1600) quand le grep tourne en même temps.

Afin de confirmer la perturbation avec grep, j'ai fait deux mesures en laissant grep tourner derrière. Ici on peut clairement voir que le standard déviation monte jusqu'à 4000.

(1 et 2: avec grep)

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 988154.000000 (position = 867)
  Maximum = 1011953.000000 (position = 866)
  Sum      = 999998156.000000
  Mean     = 999998.156000
  Variance = 1288267.552124
  Std Dev  = 1135.018745
  CoV      = 0.001135
-----

redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 967902.000000 (position = 24)
  Maximum = 1032083.000000 (position = 23)
  Sum      = 999998922.000000
  Mean     = 999998.922000
  Variance = 16789958.226318
  Std Dev  = 4097.555152
  CoV      = 0.004098
-----
```

Je pense que le processus grep tourne sur tous les cœurs mais que parfois, il utilise le cœur utilisé par le timer. Donc, il arrive à perturber un peu le timer.

## ./cpu\_loop

Plusieurs cœurs :

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./cpu_loop & ./timer 1000 1000 | ./summary
[1] 6131
----- summary1.c -----
Total of 1000 values
  Minimum = 992779.000000 (position = 741)
  Maximum = 1007704.000000 (position = 740)
  Sum      = 999998166.000000
  Mean     = 999998.166000
  Variance = 278953.335815
  Std Dev  = 528.160332
  CoV      = 0.000528
-----

redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ Nombre d'iteration par seconde = 403 (6131)
./cpu_loop & ./timer 1000 1000 | ./summary
[2] 6134
----- summary1.c -----
Total of 1000 values
  Minimum = 992966.000000 (position = 936)
  Maximum = 1007282.000000 (position = 9)
  Sum      = 999998471.000000
  Mean     = 999998.471000
  Variance = 364446.783203
  Std Dev  = 603.694280
  CoV      = 0.000604
-----

[1] Done
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./cpu_loop
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ Nombre d'iteration par seconde = 387 (6134)
```

Bien sûr, on ne voit pas de différence car on a plusieurs cœurs. Par contre, voici ci-dessous deux mesures avec un cœur.

Un cœur :

```
taskset -pc 0 $$
pid 2630's current affinity list: 0-7
pid 2630's new affinity list: 0
[2]+ Done ./cpu_loop
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./cpu_loop & ./timer 1000 1000 | ./summary
[1] 6149
----- summary1.c -----
Total of 1000 values
  Minimum = 848948.000000 (position = 451)
  Maximum = 36141081.000000 (position = 25)
  Sum      = 1843988324.000000
  Mean     = 1843988.324000
  Variance = 9554331684501.316406
  Std Dev  = 3091008.198711
  CoV      = 1.676262
-----
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ Nombre d'iteration par seconde = 185 (6149)
[1]+ Done ./cpu_loop
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ ./cpu_loop & ./timer 1000 1000 | ./summary
[1] 6152
----- summary1.c -----
Total of 1000 values
  Minimum = 861671.000000 (position = 945)
  Maximum = 28117029.000000 (position = 25)
  Sum      = 1847988038.000000
  Mean     = 1847988.038000
  Variance = 9276865320028.994141
  Std Dev  = 3045794.694333
  CoV      = 1.648168
-----
```

On peut voir que les résultats sont complètement à la masse. Ce qui est normal car le processus `cpu_loop` prend beaucoup de temps processeur. En effet, comme ils travaillent sur le même cœur, il arrive que le timer se fasse pré-empter à des moments critiques.

## Ping

Voici 2 mesures en recevant des pings:

```
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 995397.000000 (position = 3)
  Maximum = 1005510.000000 (position = 1)
  Sum      = 999997410.000000
  Mean     = 999997.410000
  Variance = 197116.401367
  Std Dev  = 443.977929
  CoV      = 0.000444
-----
redsuser@a07pc08:/cours_REDS/PTR/IsaiaSpinelli/Labo2/code$ sudo ./timer 1000 1000 | ./summary
----- summary1.c -----
Total of 1000 values
  Minimum = 987786.000000 (position = 670)
  Maximum = 1011231.000000 (position = 669)
  Sum      = 999989397.000000
  Mean     = 999989.397000
  Variance = 1001287.337891
  Std Dev  = 1000.643462
  CoV      = 0.001001
-----
```

On peut voir que la variance est comme d'habitude, elle peut varier mais rien de très spécial.

## Conclusion

Afin de perturber notre timer, il faut prendre beaucoup de temps CPU sur le même cœur où le timer tourne. Sinon, on peut voir que le timer est en moyenne précis avec une moyenne qui s'écarte de 2 ou 3 ns du temps souhaité.