Université Catholique de Louvain

LINFO1252

Système Informatique : Projet 1

Bourez Nicolas 2846-20-00
Sonnet Thibaut 6872-19-00

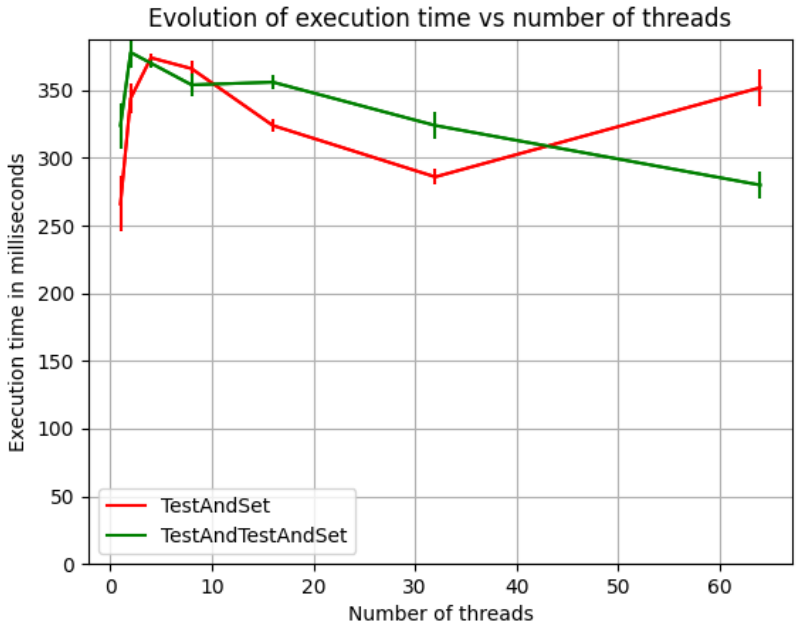
10 juin 2025

1 Introduction

L'objectif de ce rapport est de comparer l'exécution de programmes multithreadés utilisant des sémaphores `POSIX` ou des mécanismes d'attente active `Test-And-Test-And-Set` et `Test-And-Set`. Pour cela, trois problèmes célèbrent ont été implémentés à savoir le problème du producteur-consommateur, celui des philosophes et celui du lecteur-écrivain. Les mesures ont été effectués à l'aide d'un script permettant de faire varier le nombre de threads entre 2 le double de coeurs de notre machine. Une machine de 32 coeurs ayant été mise à notre disposition, le nombre de threads varie jusqu'à 64.

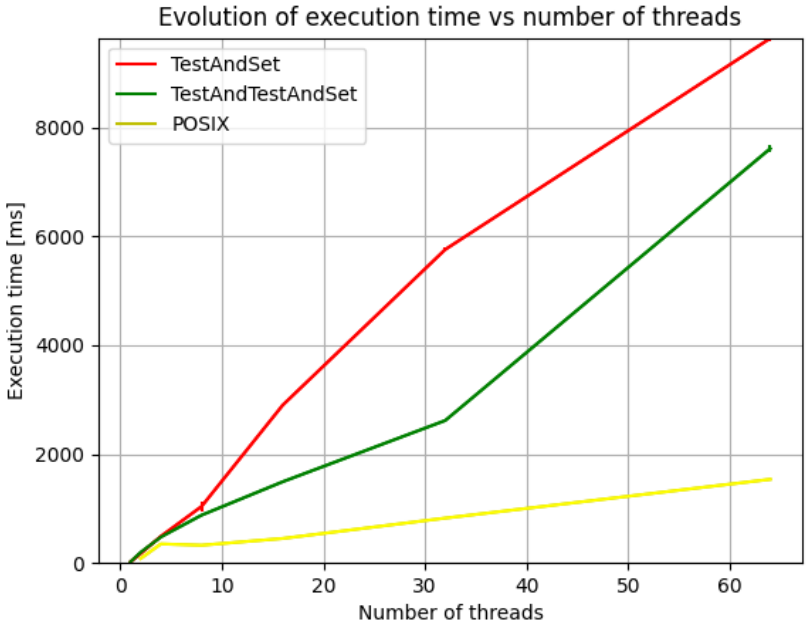
A noter que les valeurs des variances ont été divisées par 5 sur nos graphes afin d'obtenir une meilleure visibilité.

2 Analyse de Test-and-Set et Test-And-Test-And-Set



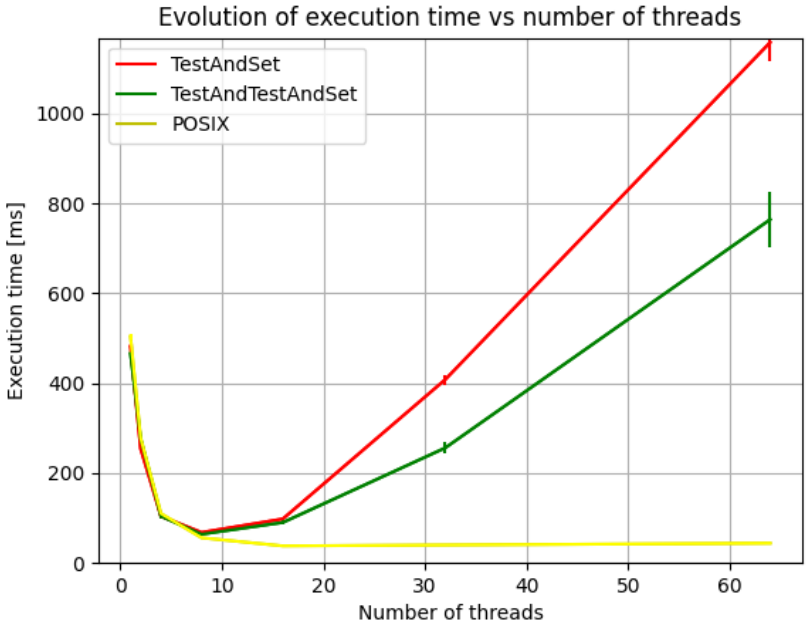
Comme on peut le voir graphiquement, les deux méthodes nous donnent quasiment le même résultat. On peut néanmoins remarquer que plus le nombre de threads augmente, plus la primitive d'attentes `Test-And-Test-And-Set` gagne du temps par rapport à `Test-And-Set`. Cela peut s'expliquer par le fait que pus le nombre de threads est important plus le nombre d'appels aux interfaces lock et unlock simultanées sont importantes et donc plus la différence devient flagrante entre les deux primitives d'attentes. Néanmoins cette justification n'explique pas la différence quand on a trente-deux threads. En effet, si on suit le raisonnement énoncé plus haut on pourrait croire que quand les trente-deux threads sont utilisés la primitive active test and test and set devrait être plus rapide mais ce n'est pas le cas. Nous pensons que cela est du au fait que certains threads ont le temps de finir leur exécution avant même que d'autres threads ne commencent la leur et donc cela diminue le nombre de threads essayant de faire une exclusion mutuelle simultanément.

3 Problème des philosophes



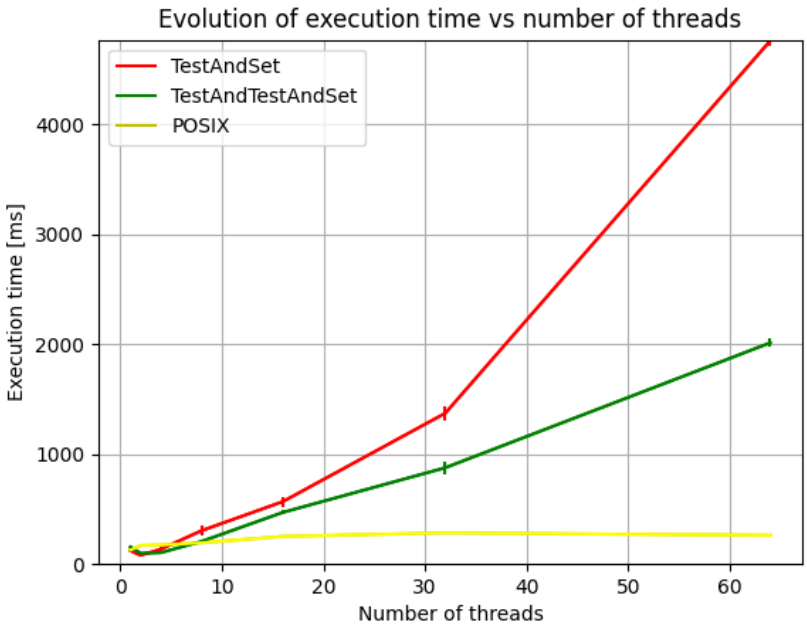
Comme on peut le voir graphiquement et comme attendu, l'exécution la plus rapide est celle qui utilise les `POSIX` la suivante est celle qui utilise `test-And-Test-And-Set` et finalement celle qui utilise `Test-And-Set`. Pour ce qui est de l'évolution général du graphique, on peut voir que plus le nombre de philosophes est élevé plus le temps d'exécution augmente. C'est logique puisque chaque philosophe doit manger 100 000 fois et que le système d'exclusion mutuelle n'est pas parfait ce qui rajoute du temps. On peut imaginer que au-delà de trente-deux threads, le temps d'exécution par philosophe va légèrement diminuer puisqu'il y aura plus de baguette que de de philosophes actifs et donc le système sera moins sensible aux exclusions mutuelles. C'est peut être ce phénomène qui explique le rétrécissement de l'écart de temps entre les mesures à soixante-quatre philosophes pour `Test-And-Set` et `Test-And-Test-And-Set`.

4 Problème du producteur et consommateur



Théoriquement , les mécanismes d'attentes actives sont efficace pour de courtes sections critiques, typiquement dans le cas où nos threads partagent une structure de données. C'est bien le cas dans ce problème-ci puisque les producteurs-consommateurs partagent un tableau de données. Le nombre de tâche étant fixe (8192), plus on a de producteurs et de consommateurs, plus l'ensemble de tâches est censé s'effectuer rapidement. C'est bien le cas pour les sémaphores `POSIX` mais pas pour les attentes actives. Il y a plusieurs raisons à cela. Premièrement on remarque que jusqu'à 4 threads, les données sont relativement identiques. Etant donné que la `for` se fait à l'extérieur de la section critique, On peut faire des opérations en parallèle et ainsi diminuer notre temps d'exécution. Cela s'explique aussi par le fait qu'il y a peu d'exclusions mutuelles entre nos threads. Cependant, plus le nombre de threads augmente, plus la différence entre les sémpahores `POSIX` et les sémpahores d'attente active est importante. Notre implémentation reste donc moins efficace que des threads `POSIX`, cela s'explique par le fait que le nombre d'appel à l'instruction atomique `xchgl` augmente considérablement ce qui impacte le temps d'éxecution de manière non-négligebale.

5 Lecteur écrivain



De manière générale, pour les 3 types de sémaphores, le temps d'exécution augmente en fonction du nombre de threads. C'est normal car la section critique prend énormément de temps et c'est dans celle-ci que se déroule la fonction principale de nos threads. La différence entre le `Test-And-Set` , le `Test-And-Test-And-Set` et les sémpahores `POSIX` est dû à l'instruction atomique `xchgl` qui prend beaucoup de temps.

6 Conclusion

Comme il a été montré tout au long de ce rapport les différentes primitives d'attentes actives sont la plupart du temps bien moins efficaces que les `POSIX`. Malgré l'implantation `Test-And-Test-Set` qui est une amélioration de `Test-And-Set` le `POSIX` reste largement supérieur. Néanmoins pour la plupart des tests quand le nombre de threads est petit les différents temps d'exécution reste similaire voir parfois meilleur pour les primitives d'attente actives.