|  |
| --- |
| HE-Arc |
| Rapport OS TP final |
| Threads |

|  |
| --- |
| Wirth Jeremy & Rey Tom  05/05/2016 |

Table des matières

[Threads Posix 2](#_Toc450838048)

[Implémentation 2](#_Toc450838049)

[Différence avec/sans threads 2](#_Toc450838050)

[Visualisation deux threads sur une même ressource 3](#_Toc450838051)

[Visualisation deux threads de manière sécurisé 3](#_Toc450838052)

[Conclusion 4](#_Toc450838053)

# Threads Posix

Les threads POSIX, souvent appelés pthreads, sont un sous-standard de la norme POSIX décrivant une interface de programmation permettant de gérer des threads.

Un thread ou fil (d'exécution) est similaire à un [processus](https://fr.wikipedia.org/wiki/Processus_(informatique)) car tous deux représentent l'exécution d'un ensemble d'instructions du [langage machine](https://fr.wikipedia.org/wiki/Langage_machine) d'un [processeur](https://fr.wikipedia.org/wiki/Processeur). Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en [parallèle](https://fr.wikipedia.org/wiki/Parall%C3%A9lisme_(informatique)). Toutefois, là où chaque processus possède sa propre [mémoire virtuelle](https://fr.wikipedia.org/wiki/M%C3%A9moire_virtuelle), les threads d'un même processus se partagent sa mémoire virtuelle. Par contre, tous les threads possèdent leur propre [pile d’appel](https://fr.wikipedia.org/wiki/Pile_d%27ex%C3%A9cution).

|  |  |
| --- | --- |
| **Routine Prefix** | **Functional Group** |
| **pthread\_** | Threads themselves and miscellaneous subroutines |
| **pthread\_attr\_** | Thread attributes objects |
| **pthread\_mutex\_** | Mutexes |
| **pthread\_mutexattr\_** | Mutex attributes objects. |
| **pthread\_cond\_** | Condition variables |
| **pthread\_condattr\_** | Condition attributes objects |
| **pthread\_key\_** | Thread-specific data keys |
| **pthread\_rwlock\_** | Read/write locks |
| **pthread\_barrier\_** | Synchronization barriers |

<https://computing.llnl.gov/tutorials/pthreads/>

# Implémentation

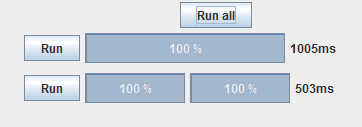
Pour notre application, nous utiliserons trois cas pour montrer de manière didactique le fonctionnement des threads et leurs utilités.

1. Différence de temps de calcule avec ou sans thread
2. Visualisation de l’occupation des threads sur une même ressource
3. Visualisation de l’occupation des threads de manière correcte avec des verrous

Chacune des parties décrites ci-dessus se retrouveront dans notre application et elles seront décrites dans ce rapport.

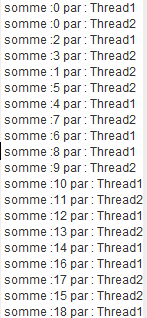
## Différence avec/sans threads

Le but étant ici de montrer qu’utiliser plusieurs threads augmentent la vitesse de calcul car les tâches sont effectuées en parallèle. On voit ici la différence entre un seul thread et deux threads. On remarque que la différence de temps est du simple au double. Cet écart paraît logique car on utilise deux threads, donc deux tâches en parallèle donnent un gain de temps deux fois supérieur.



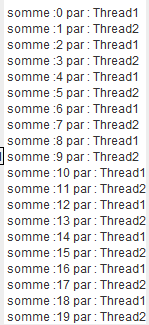
## Visualisation deux threads sur une même ressource

Le but ici est de voir l’utilisation d’une même ressource par deux threads. On constate que les choses ne se déroulent pas simplement comme une boucle.



## Visualisation deux threads de manière sécurisé

Utilisation de mutex pour bloquer l’appel à la méthode d’addition tant qu’un thread travaille à l’intérieur de celle-ci. Avec la libération de la ressource une fois que le thread à finit de travailler.



# Conclusion

Avec ces trois exemples, on remarque l’utilité des threads, notamment en termes de vitesse de calcul. Cependant, on voit que l’utilisation de threads peut amener à des comportements imprévisibles comme dans l’exemple 2. Il est donc nécessaire de gérer des éléments supplémentaires lorsqu’on utilise des threads comme l’utilisation de mutex pour limiter l’accès à une ressource.