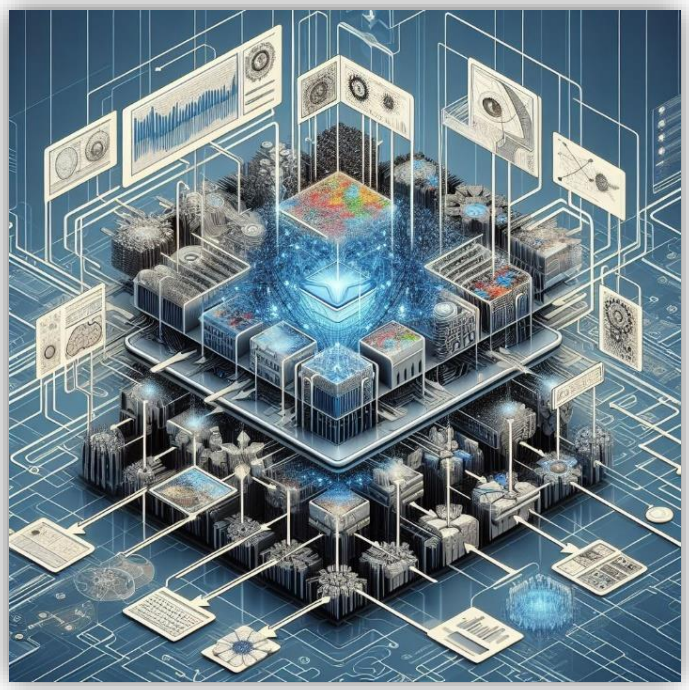

Sommaire

Exercice 1

Dans ce programme, nous avons 3 threads différents :

- Le thread principal
- Le thread first qui s'occupe de la fonction *foo*
- Le thread second qui s'occupe de la fonction *bar*

Exercice 2

Nombre de threads	**Moyenne du temps de calcul sur 10 exécutions (ms)**
2	1723
4	1012
8	698
16	589

J'ai remarqué que le temps de calcul baissait quand j'augmentais le nombre de thread, ce qui était cohérent car je partageais le calcul entre les threads. En mesurant le temps de calcul moyen qui est de 1006 ms avec un écart type de 408 ms, ce qui indiquait une grande variabilité des résultats.

L'écart type valait environ la moitié de la moyenne. J'ai constaté que 4 threads offraient un temps de calcul moyen, tandis que 2 threads étaient nettement plus lents. A l'inverse, 8 et 16 threads étaient beaucoup plus rapides que la moyenne.

Exercice 3

Dans ce programme, chaque thread calcule une partie de la somme de pi et l'ajoute à une variable globale qui contient le résultat final. Cela peut créer un problème de concurrence si deux threads ou plus tentent d'écrire dans la variable globale en même temps.

Pour éviter ce problème, j'ai utilisé un mutex pour protéger l'accès à la variable globale. Ainsi, chaque thread doit acquérir le mutex avant d'écrire dans la variable globale, et le relâcher après. Cela garantit qu'un seul thread à la fois peut modifier la variable globale.

Exercice 4

Dans ce programme, le résultat du calcul de la somme de pi est stocké dans un moniteur qui assure l'exclusion mutuelle sur son accès. Ce moniteur dispose de deux méthodes, une pour ajouter une valeur au résultat et une pour le lire. On peut comparer ce moniteur à un problème de lecteur/rédacteur, sauf qu'ici il n'y a qu'un seul lecteur qui est le thread principal et plusieurs rédacteurs qui sont les threads qui calculent la somme partielle. Le lecteur n'a pas besoin d'entrer en section critique pour lire le résultat, car il le fait seulement après avoir synchronisé tous les rédacteurs et qu'ils ne modifient donc plus la variable.

Exercice 5

Dans ce programme, l'utilisation de 2 moniteurs est nécessaire. Le premier moniteur est utilisé pour conserver les paires de nombres premiers "jumeaux", tandis que le second moniteur est chargé de distribuer les nombres à calculer à chaque thread. Le fonctionnement du premier moniteur est similaire à celui du moniteur de l'exercice précédent, tandis que le second moniteur dispose d'un accesseur qui attribue la valeur à traiter à chaque thread (la valeur min donnée en entrée du programme) et un autre pour la valeur maximum.