Programmation système avancée
TP n° 6 : Mémoire partagée

Exercice 1 :

Nous vous fournissons un programme `quicksort.c` qui trie une liste de nombres à l'aide de l'algorithme *quicksort*.

1. Mesurez les performances du programme `quicksort.c`, et notez le résultat.
2. Modifiez le programme pour que le tableau à trier se trouve dans une zone de mémoire anonyme partagée allouée à l'aide de `mmap`. Les performances ont-elles changé ?
3. Modifiez le programme pour que, après la première partition, il crée deux fils à l'aide de `fork` puis trie les deux demi-tableaux dans les deux processus ; il attendra ensuite la fin de l'exécution des fils avant de continuer. Pourquoi n'y a-t-il pas besoin de primitives de synchronisation ? Mesurez les performances de votre programme.
4. Modifiez votre programme pour qu'il utilise 2^n processus, où n est passé en ligne de commande. Quelle est la valeur de n qui donne les meilleures performances ?
5. Comment faire pour utiliser un nombre de processus qui n'est pas une puissance de 2 ?

Exercice 2 :

Ératosthène écrit les entiers de 2 à 100 dans la cour de la Bibliothèque d'Alexandrie (voir figure 1). Il voit que la première valeur est 2, il envoie un esclave barrer tous les multiples non-triviaux de 2 (4, 6, 8 etc.). Il voit que la première valeur qui reste est 3, il envoie un esclave barrer tous les multiples non-triviaux de 3. Il voit que la première valeur qui reste est 5, il envoie un esclave barrer tous les multiples non-triviaux de 5, et ainsi de suite. Les entiers qui restent à la fin sont exactement les nombres premiers compris entre 2 et 100.

2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3		5		7		9		11		13		15
2	**3**		5		7				11		13		

FIGURE 1 – Le crible d'Eratosthène

1. Écrivez un programme qui utilise l'algorithme du crible d'Ératosthène pour compter le nombre de nombres premiers compris entre 2 et n, où n est un argument de ligne de commande. Votre programme ne devra pas faire de multiplications à l'intérieur de la boucle interne. La liste de nombres sera implémentée par un tableau de `char` dont chacun vaut 1 si l'entier correspondant n'a pas encore été barré, et 0 sinon. Vérifiez que votre programme retourne 25 pour un paramètre valant 100, et 168 pour un paramètre valant 1000.
2. La mémoire n'est pas forcément cohérente, il n'est donc pas correct de paralléliser le crible sans utiliser de primitives de synchronisation ou d'opérations atomiques. Modifiez votre programme pour qu'il manipule un tableau de `atomic_char` (défini dans `<stdatomic.h>`) et les fonctions `atomic_load` et `atomic_store`. Les performances ont-elle changé ?
3. Modifiez votre programme pour qu'il alloue le tableau dans une zone de mémoire partagée, puis qu'il engendre un fils pour chaque entier dont il faut barrer les multiples. Que pensez-vous des performances du programme ainsi obtenu ? Expliquez d'où vient le problème.