Ibrahim RIAZI

Fahed imed ZIDANI **Groupe1**

TP 4 : Multi-clients : Pool de thread

Serveur simple

Si deux clients demandent le même port, seule la première demande aboutira.

Solutions ?

Quelles solutions connaissez-vous pour gérer les entrées/sorties concurrentes et asynchrones ?

Réponse :

- Les threads

Test de performance 1 :

Comparaison du comportement de notre serveur pour les valeurs de n :

Pour les valeurs (1, 2, 10, 100, 1000) notre serveur reçoit toutes les lignes enviées par les clients stressants, mais pour la valeurs 5000 le serveur ne reçoit pas toutes les lignes et on eu une erreur : **SocketException: Too many open files** .

Oui il y a une différence de comportement de notre serveur après la fermeture de la connexion immédiatement, notre serveur reçoit toutes les lignes enviées.

Performance temporelle :

→ Les fichiers .CSV

Pour le stress2.java on a écrit le code mais on a toujours une erreur qu'on n'arrive pas à résoudre (voir code) .

Pool de Threads

Executors.newCachedThreadPool.

Il existe deux signatures pour cette méthode sont :

- **newCachedThreadPool(ThreadFactory threadFactory) :** Crée un pool de threads qui crée de nouveaux threads selon les besoins, et utilise la ThreadFactory fournie pour créer de nouveaux threads si nécessaire.

- **newCachedThreadPool() :** Si aucun thread existant n'est disponible, un nouveau thread sera créé et ajouté au pool. Les threads qui n'ont pas été utilisés pendant soixante secondes sont terminés et supprimés du cache (Le nombre de threads est dynamique).

Executors. newFixedThreadPool.

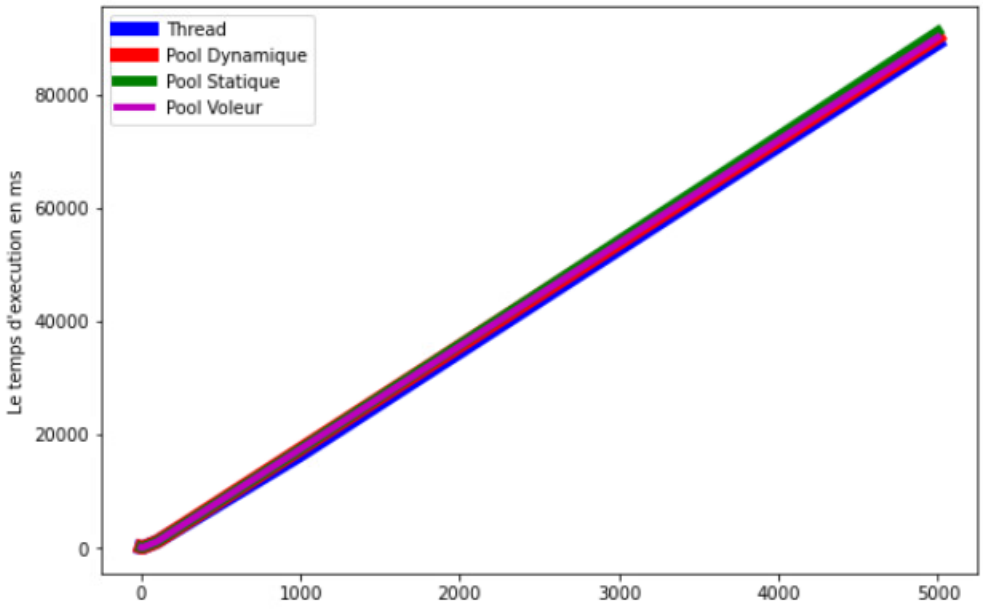
Il existe deux signatures pour cette méthode sont :

- **newFixedThreadPool(int nbThreads) :** Crée un pool de threads qui réutilise un nombre fixe de threads fonctionnant sur une file d'attente illimitée partagée.
- **newFixedThreadPool(int nThreads, ThreadFactory threadFactory) :** Crée un pool de threads qui réutilise un nombre fixe de threads fonctionnant sur une file d'attente illimitée partagée, en utilisant la ThreadFactory fournie pour créer de nouveaux threads si nécessaire.

Test de performance 2/3/4 :

On a constaté que les pools dynamiques sont plus rapide que les statiques et les voleurs sont plus rapide que les dynamiques et les statiques.

Bilan :



- **les pools dynamiques plus rapide que les statiques**
- **les voleurs sont plus rapide que les dynamiques et les statiques.**
- **les threads et le pools ont le même temps d'exécution a peu près et l'avantage avec les pools c'est qu'ils minimisent la consommation des ressources .**