Rapport Labo 6

Ali Zoubir & Kenan Augsburger

January 14, 2025

Contents

1. Informations sur le projet

1.1. Répertoire Git (sur GitHub)

mondotosz/pco_lab06°

1.2. Membres

- Ali Zoubir°
- Kénan Augsburger°

1.3. Introduction du problème

Dans ce laboratoire, l'objectif était de concevoir et implémenter un pool de threads capable de gérer efficacement l'exécution de tâches concurrentes tout en respectant les contraintes définies : limitation des threads actifs, gestion de la surcharge via une file d'attente limitée, et recyclage des threads inactifs après un délai donné.

2. Implémentation et choix

2.1. Architecture des classes

La classe principale `ThreadPool` repose sur un moniteur de Hoare pour gérer la synchronisation. Les threads sont encapsulés dans une structure qui contient l'état de chaque thread (actif, en attente, ou expiré).

Pseudocode pour `ThreadPool` :

```pseudocode
Classe ThreadPool :
  initialiser(max_threads, max_queue, timeout) :
    // Préparer les structures pour la gestion des threads

  démarrer(tâche) :
    si (file pleine) alors rejeter(tâche)
    sinon assigner_à_thread(tâche)

  boucle_thread() :
    tant que (actif) :
      si (délai dépassé) alors terminer_thread()
      sinon exécuter_tâche()
```

2.2. Méthodes principales

- `start` : Gère l'ajout de tâches. Si un thread est disponible, il exécute immédiatement la tâche. Sinon, la tâche est mise en file d'attente ou rejetée si la file est pleine.
- `worker_loop` : Chaque thread attend une tâche ou termine après un délai d'inactivité.

Pseudocode simplifié de la logique :

```pseudocode
si (file non vide) alors :
  tâche = retirer_file()
  exécuter(tâche)
sinon si (temps_inactivité atteint) :
  terminer_thread()
```

2.3. Subtilités concurrentielles

L'implémentation évite les blocages grâce à des conditions de synchronisation, garantissant un accès sûr aux données partagées. La destruction du pool est gérée proprement pour éviter les fuites de threads.

3. Tests

3.1. Scénarios de test

Les tests incluent :

1. Gestion de 10 tâches avec 10 threads.
2. Surcharge de la file d'attente.
3. Recyclage des threads inactifs.

Pseudocode pour un test :

```pseudocode
pool = initialiser_pool(10, 50, 100ms)
envoyer_tâches(pool, 10)
vérifier(toutes_tâches_finies)
```

3.2. Résultats

Les tests montrent que le pool respecte les limites et fonctionne efficacement dans des scénarios variés.

4. Remarques et conclusion

4.1. Améliorations possibles

- Réduction des ressources consommées par les threads inactifs.
- Ajout de métriques pour surveiller les performances.

4.2. Conclusion

Ce projet illustre les principes de la programmation concurrente, offrant une solution robuste pour gérer les tâches dans des environnements à forte charge.