3259.1 Paradigmes de program. avancés II– Rapport technique – ISC3il-b

|  |
| --- |
| **Laboratoire 2** |

|  |
| --- |
| Étudiants participant à ce travail :  **Nicolas Aubert, ISC3il-b**  **Théo Vuilliomenet, ISC3il-b**  Présenté à :  **Aïcha Rizzotti**  **Julien Senn**  Restitution du rapport : **12.05.2023**  Période : **2023**  École : **HE-Arc, Neuchâtel** |

**Mesure de performances d’un modèle producteur / consommateur en Java**

Table des matières

[1 - Glossaire 2](#_Toc133572298)

[2 - Introduction 3](#_Toc133572299)

[2.1 - Contexte 3](#_Toc133572300)

[3 - Présentation du framework 4](#_Toc133572301)

[3.1 - Classe Buffer 4](#_Toc133572302)

[3.2 - Classe Producer 4](#_Toc133572303)

[3.3 - Classe Consumer 4](#_Toc133572304)

[4 - Résultats 5](#_Toc133572305)

[4.1 - 20 producteurs, 20 consommateurs, buffer de taille 5 5](#_Toc133572306)

[4.2 - 100 producteurs, 200 consommateurs, buffer de taille 50 5](#_Toc133572307)

[5 - Limitations et perspectives 6](#_Toc133572308)

[6 - Conclusion 7](#_Toc133572309)

[7 - Annexes I](#_Toc133572310)

[7.1 - Table des illustrations I](#_Toc133572311)

[7.2 - Bibliographies et références II](#_Toc133572312)

[7.2.1 - Sites Web II](#_Toc133572313)

1. Glossaire

|  |  |
| --- | --- |
| **Buffer** | Zone de stockage temporaire pour les données entre un producteur et un consommateur. |
| **Concurrence** | Situation où plusieurs processus ou threads sont en compétition pour l'accès aux ressources d'un système. |
| **Débit** | Nombre d'actions exécutées, de résultats d'actions exécutées ou de résultats produits par unité de temps. |
| **Latence** | Temps nécessaire pour effectuer une action ou produire un résultat. |
| **Log** | Enregistrement des actions qui se produisent dans le programme. |
| **Modèle producteur / consommateur** | Modèle de communication entre deux parties où l'une produit des données et l'autre les consomme. |
| **Opération atomique** | Une opération est atomique si elle est exécutée entièrement ou pas du tout, sans être interrompue par d'autres opérations en cours d'exécution sur d'autres threads. Une opération atomique garantit l'intégrité des données, même en présence de threads concurrents. |
| **Performances** | Mesure de la quantité de travail qu'une application peut traiter dans une période donnée et de la vitesse à laquelle elle peut traiter une unité de travail. |
| **Thread** | Unité d'exécution dans un programme qui permet une exécution parallèle de plusieurs tâches. |
| **Thread safe** | Une classe ou une méthode est thread safe si elle peut être utilisée simultanément par plusieurs threads sans causer de problèmes de concurrence ou d'incohérence de données. |

1. Introduction
   1. Contexte

L'objectif de ce projet est de réaliser un programme de monitoring de la concurrence en utilisant le modèle producteur-consommateur. Nous allons nous intéresser à la mise en place d'un framework qui nous fournira des informations utiles sur les performances d'un programme.

Pour ce faire, nous implémenterons un exemple simple du modèle producteurs-consommateurs, en Java, en prenant en compte des paramètres importants, tels que :

* La taille du buffer,
* Le nombre de producteurs,
* & le nombre de consommateurs.

1. Présentation du framework
   1. Classe Buffer

La classe Buffer est une implémentation du modèle producteur-consommateur ; elle est utilisée pour stocker des données produites par un producteur jusqu'à ce qu'elles soient prêtes à être consommées par un consommateur. Le fonctionnement de cette classe repose sur l'utilisation d'une blocking queue, qui est une structure de données qui permet aux threads de se bloquer lorsqu'ils essaient d'ajouter ou de retirer des éléments à partir de la queue vide ou pleine. Une blockingqueue est également thread safe, cela signifie qu’une ressource peut être uniquement modifiée de façon atomique.

Le constructeur de la classe prend en entrée la taille du buffer, qui est utilisée pour initialiser la blocking queue. Si aucune taille n'est spécifiée, la taille par défaut est de 100.

La méthode push permet d'ajouter des données au buffer. Cette méthode utilise la méthode put de la blocking queue, qui bloque le thread si la queue est pleine jusqu'à ce qu'il y ait de la place pour ajouter la donnée.

La méthode take permet de retirer des données du buffer. Cette méthode utilise la méthode take de la blocking queue, qui bloque le thread si la queue est vide jusqu'à ce qu'une donnée soit disponible pour être retirée.

La méthode getSize permet d'obtenir la taille du buffer.

* 1. Classe Producer
  2. Classe Consumer

1. Résultats
   1. 20 producteurs, 20 consommateurs, buffer de taille 5
   2. 100 producteurs, 200 consommateurs, buffer de taille 50
2. Limitations et perspectives

Le projet présenté dans ce rapport comporte certaines limites qui peuvent être adressées pour améliorer sa pertinence et son efficacité. Tout d'abord, l'absence d'interface graphique peut rendre l'utilisation de l'application difficile pour les néophytes. Il pourrait être envisagé de développer une interface graphique plus conviviale en utilisant des outils tels que Swing, afin de faciliter l'interaction avec l'application.

En outre, une limitation actuelle est que le type de buffer utilisé est fixe et ne peut pas être modifié. Cela peut limiter la flexibilité de l'application dans certaines situations. Il serait bénéfique de permettre à l'utilisateur de choisir le type de stockage approprié en fonction des besoins spécifiques de son projet.

Une autre limitation est que l'application actuelle ne prend pas en charge les variantes du modèle producteur / consommateur telles que la priorité aux producteurs ou la priorité aux consommateurs. L'ajout de ces fonctionnalités permettrait de rendre l'application plus polyvalente et plus adaptée aux différents scénarios de producteurs / consommateurs.

Enfin, il est important de noter que JConsole n'a pas été utilisé dans le cadre de ce projet, mais il pourrait être intéressant de l'intégrer à l'avenir. En effet, JConsole permet de collecter des informations détaillées sur les performances de la JVM, telles que la consommation de mémoire, l'utilisation du processeur et bien plus encore. En utilisant JConsole, il serait possible de surveiller l'application de manière plus précise et d'optimiser les performances de manière plus fine. Cette amélioration pourrait être envisagée dans le cadre d'une perspective future pour le projet.

1. Conclusion

* Framework fonctionnel,
* Possibilité de mesurer les performances d’un modèle producteur / consommateur en Java
* Plusieurs paramètres peuvent être définis au lancement du programme, tels que :
  + Nombre de producteurs,
  + Nombre de consommateurs,
  + Taille du buffer
  + Nombre d’action à effectuer par les producteurs / consommateurs
* ??

1. Annexes
   1. Table des illustrations

**Aucune entrée de table d'illustration n'a été trouvée.**

* 1. Bibliographies et références
     1. Sites Web

Using JConsole - Java SE Monitoring and Management

Guide. (2006, October 1). <https://docs.oracle.com/javase/8/docs/technotes/guides/management/jconsole.html>

Baeldung. (2021, December 21). *Guide to java.util.concurrent.BlockingQueue*. Baeldung. <https://www.baeldung.com/java-blocking-queue>

*Executor (Java Platform SE 8 )*. (2023, April 5). <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html>

Baeldung. (2022, November 24). *A Guide to the Java ExecutorService*. Baeldung. <https://www.baeldung.com/java-executor-service-tutorial>