



Projet Données Réparties

Rapport

Enzo PETIT

Nam VU

16 janvier 2022

ENSEEIHT – 2SN-A

Table des matières

1	Introduction	3
2	Version en mémoire partagée	3
2.1	Réalisation	3
2.1.1	tryTake, tryRead	3
2.1.2	takeAll, readAll	3
2.1.3	eventRegister	4
2.1.4	write	4
2.1.5	take, read	4
2.2	Tests	4
3	Version client / mono-serveur	5
4	Applications	5
4.1	Calcul des nombres premiers	5
4.2	Recherche approximative dans un fichier	5
5	Conclusion	5

1 Introduction

Le projet Linda a pour but de réaliser un espace partagé de données typées.

Une version à mémoire partagée avec gestion de callback a été réalisée ainsi qu'une version client-serveur en RMI se reposant sur cette dernière.

Pour la première version, la classe `linda.shm.CentralizedLinda` a ainsi été complétée et des tests unitaires sous JUnit 5 ont été rédigés dans `linda.test.CentralizedLindaTest`.

La version client-serveur se trouve quand à elle dans le package `linda.server` et utilise la version *shm* inchangée pour la partie serveur.

Enfin quelques applications utilisant Linda ont été réalisées telle que la recherche de nombre premiers selon l'algorithme du crible d'Erathosthène, en séquentiel et parallèle, ainsi que la parallélisation de l'application de recherche.

2 Version en mémoire partagée

Cette section reprend le rapport provisoire de décembre dernier.

2.1 Réalisation

A l'instanciation, `CentralizedLinda` initialise trois tableaux pour le stockage des tuples (`tupleSpace`) et les events *take* (`takeEvents`) et *read* (`readEvents`).

Ces tableaux sont de type `CopyOnWriteArrayList` qui est une variante *thread-safe* de l'`ArrayList` classique adaptée à un contexte concurrent où le nombre de lectures est bien supérieure au nombre d'écritures.

Suivent après les détails d'implémentation des différentes opérations, plus ou moins dans l'ordre de réalisation :

2.1.1 tryTake, tryRead

Ces deux méthodes sont non bloquantes, on itère simplement sur la liste (en partant de la tête) et on renvoie le premier tuple (le plus vieux) qui match le template. `null` est renvoyé si aucun tuple actuellement stocké ne correspond.

2.1.2 takeAll, readAll

Même chose que précédemment mais on stocke tous les tuples correspondants dans une `ArrayList` que l'on renvoie à la fin (qui est vide si aucun résultat).

2.1.3 eventRegister

En commençant à vouloir implémenter les **take** et **read** bloquant on s'est demandé comment pouvait-on "proprement" et avec le moins d'effort possible bloquer et débloquer les appels : le principe des event nous a paru bien adapté pour réaliser cette tâche (détails plus loin).

En mode **IMMEDIATE** un tuple est retourné immédiatement dans le callback en cas de match sur l'espace actuel (via **tryTake/tryRead**), sinon on range l'événement en attente dans le tableau correspondant (**takeEvents** ou **readEvents**).

2.1.4 write

La méthode **write** étant la "porte d'entrée" de tous les tuples vers l'espace de stockage de Linda, c'est là qu'on en profite pour "résoudre" les événements en attente le cas échéant.

Ainsi on itère d'abord sur les *read* en attente (**readEvents**), vérifie si le tuple à écrire "match" le template de l'événement et le cas échéant on appelle le callback correspondant.

Ensuite on fait de même avec les *take* en attente (**takeEvents**) mais au premier match (du plus vieux), on résout le callback et on retourne, immédiatement. Le tuple n'est pas enregistré et les *take* en attente dessus mais plus récents attendront le prochain tuple correspondant.

Finalement si aucun *take* n'attendait le tuple, on le sauvegarde dans **tupleSpace**.

Un tuple en entrée peut ainsi résoudre tous les *read* en attente mais qu'un seul *take* en attente, le plus vieux.

2.1.5 take, read

Un *take* ou *read* bloquant revient à enregistrer un événement *immédiat* dont le callback renvoie le tuple passé en entrée, rester bloqué jusqu'à résolution de celui-ci et finalement renvoyer son résultat.

On utilise pour faire ça une **LinkedBlockingQueue**, queue bloquante : le callback de l'événement correspond à la méthode **offer** de la queue (dépôt non bloquant) qui sera éventuellement appelée lors d'un *write*.

Le **take/read** reste lui bloqué sur le **take** de la queue et renverra son résultat quand il sera débloqué par un dépôt dans la queue.

2.2 Tests

Tous les tests **Basic** fournis passent en l'état.

Une classe de tests unitaires JUnit 5 **linda.test.CentralizedLindaTest** a aussi été écrite.

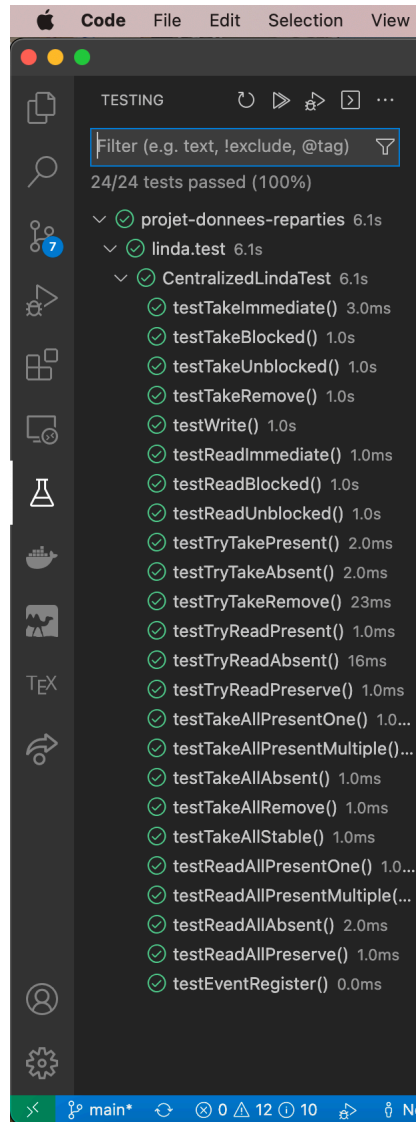


FIGURE 1 – Résultats des tests définis dans `linda.test.CentralizedLindaTest`
(Visual Studio Code + Extension Pack for Java)

3 Version client / mono-serveur

4 Applications

4.1 Calcul des nombres premiers

4.2 Recherche approximative dans un fichier

5 Conclusion