École nationale supérieure d'électrotechnique, d'électronique, d'informatique,
d'hydraulique et des télécommunications
Filière Informatique et Mathématiques appliquées

Rapport du projet de systèmes concurrents / intergiciels

Olivier Lienhard
Tom Lucas
Thibault Hilaire

Toulouse, le 13 Janvier 2015

Sommaire

Partie 1

Choix de la spécification libérale

Explications des différents choix de la spécification libérale :

- quand plusieurs tuples correspondent, take retourne le premier à avoir été écrit dans la mémoire (FIFO)
- quand plusieurs take sont en attente et qu'un dépôt peut en débloquer plusieurs, on débloque le premier take à avoir demandé (FIFO)
- quand des read et un take sont en attente, et qu'un dépôt peut les débloquer, on les débloque dans l'ordre de demande (FIFO)
- quand il y a un take et un callback enregistré pour le même motif, le take est prioritaire

Partie 2

Version en mémoire partagée

2.1 Choix d'implémentation

Voici les différentes structures choisies pour respecter nos choix de spécification:

- Collection<Tuple> tuples : pour sauvegarder nos tuples qui ont été écrits dans la mémoire, et les enlever lors d'un take
- Map<Tuple, LinkedList<Integer» MatchEnAttente : cette map va permettre de stocker tous les take/read bloquants
- int id : un entier qui nous permettra de simuler notre FIFO, de savoir dans quels ordres les take/read ont été écrits dans notre map
- Condition[] classe : un ensemble de conditions liées à un Lock, dont on associera chaque condition à un take/read bloquants
- Condition writeCondition : une condition qui va nous permettre de faire une priorité au signalé lors du réveil d'un take/read
- Boolean takeEffectue : un booléen permettant de vérifier, lors d'un write, si un take a été effectué

A partir de ceci, on peut expliquer l'algorithme de la fonction write :

Tout d'abord on ajoute le tuple dans la collection de tuples. Ensuite on récupère les templates dont le tuple ajouté correspond (i.e. tuple.matches(template)==true), ce qu'on appelle les templatesCorrespondants. Puis tant que cette collection de templatesCorrespondants n'est pas vide ET qu'un take (sur ce tuple ajouté) n'a pas été effectué (vérifiable sur notre booléen takeEffectue), on réveille le premier template correspondant (celui dont l'indice id est le plus bas). Si un take est effectué, le booléen takeEffectue devient vrai, on sort de la boucle et on ne réveille pas les callback et la fonction write est finie. Sinon, si aucun take

n'est effectué et si on a parcouru toute la liste des templatesCorrespondants, on notifie les callback qui sont des observateurs. Un tel algorithme garantit alors le respect de la spécification ci-dessus, qui est ensuite validée par les tests décrits ci-dessous.

2.2 Tests

Pour les tests, outre les tests donnés, des tests unitaires sont effectués pour chaque fonction pour chaque type de paramètre différent (notamment pour les callback). Ainsi une liste non exhaustive des tests unitaires principaux est :

- BasicTestAsyncCallbackReadFuture.java
- BasicTestAsyncCallbackReadImmediate.java
- BasicTestAsyncCallbackTakeFuture.java
- BasicTestAsyncCallbackTakeImmediate.java
- BasicTestCallbackReadFuture.java
- BasicTestCallbackReadImmediate.java
- BasicTestCallbackTakeFuture.java
- BasicTestCallbackTakeImmediate.java
- BasicTestRead.java

Puis, pour vérifier que la spécification est bien respectée, nous avons effectué des tests pour vérifier chacun des quatre points, nommés respectivement dans l'ordre :

- BasicTestTakeSpec1.java
- BasicTestTakeSpec2.java
- BasicTestTakeReadSpec3.java
- BasicTestTakeCallbackSpec4.java

Ces tests-là pour les principaux, ajoutés à d'autres tests, donnant tous des résultats cohérents, nous permettent de valider notre implémentation. Il est à noter que certains tests finissent sur des commandes bloquantes et il faut donc arrêter l'éxécution manuellement, cela est bien sûr fait exprès de manière à montrer dans certains cas le bon fonctionnement de nos fonctions.

Partie 3

Version client / mono-serveur

3.1 Choix d'implémentation

Partie 4

Version multi-serveurs

4.1 Choix d'implémentation