Mini-projet en Programmation concurrente

Paul Orhon – Nicolas Bourges

LP – MiAR – Université de Nantes

11 novembre 2017

Table des matières

1 Exercice 1 : Le dîner des philosophes **2**
1.1 2
1.2 2
1.3 2

2 Exercice 2 : File de communication bloquante **2**
2.1 2
2.2 2

1 Exercice 1 : Le dîner des philosophes

1.1

Lorsqu'on exécute le programme, on constate que celui-ci se bloque à un moment de l'exécution. En regardant dans les messages du terminal, on remarque qu'un philosophe mange puis pense et ainsi de suite pendant une longue période puis un autre philosophe enchaîne. Pendant ce temps, les autres philosophes sont en attente. Avant la situation de blocage, on constate que les philosophes essaient de manger tous en même temps. Dès qu'une baguette est disponible, celle-ci est prise par un philosophe même s'il n'a pas de baguette dans l'autre main. Ainsi, on se retrouve dans une situation de deadlock.

1.2

En effet, on remarque une situation de deadlock, c'est-à-dire que des threads attendent entre eux la libération de ressources avant de continuer leur exécution. Dans notre cas, les philosophes attendent que les baguettes de leurs gauches et de leurs droites se soient plus occupées pour pouvoir manger. Or, chaque philosophe se retrouve avec une baguette de disponible.

Pour résoudre ce problème, les philosophes doivent attendre leurs tours et dès que l'un a la possibilité de prendre les 2 baguettes en même temps, il mange, sinon il attend.

1.3

La solution se trouve dans le package de l'exercice 1. Lorsqu'on teste la solution, on peut voir dans le terminal le nombre de fois qu'un philosophe mange et pense. Nous avons enlevé les messages qui indiquait que le philosophe mange et pense pour alléger l'affichage du terminal.

Par ailleurs, nous avons ajouté une barre de pourcentage, pour chaque philosophe, qui représente le nombre de fois qu'il a mangé et pensé.

2 Exercice 2 : File de communication bloquante

2.1

La classe AbstractFileBloquanteBornee<E> est une classe abstraite qui permet de créer des files de communications bornées bloquantes. Les threads peuvent déposer (resp. prendre) des objets dans ces filles. Cette opération peut être bloquante si la file est pleine (resp. vide).

2.2

Pour implémenter une classe classe abstraite, il faut implémenter ces méthodes à savoir, dans notre cas, prendre() et déposer(). Ensuite, en se basant sur les descriptions de chaque méthodes, il faut mettre en place un système de gestion de thread pour éviter que les threads ajoutent un élément alors que la liste est pleine ou qu'ils essayent d'en retirer alors qu'elle est vide.