Université Paul Sabatier
31062 Toulouse Cedex 09

Master 1 Informatique - MCPR

Modèles et Concepts du Parallélisme et de la Répartition
Travaux pratiques n° 5

Threads Java et synchronisation du type « moniteur »

Exercice 1

En reprenant le principe de synchronisation déjà vu en cours et TD, programmer une application dans laquelle des threads Java, représentant `NP` producteurs et `NC` consommateurs de messages, synchronisent leurs accès à un tampon de messages géré circulairement. `NP` et `NC` peuvent varier d'une exécution à l'autre et seront donc paramètres de cette application.

Les classes `Producteur` et `Consommateur` permettront de créer les threads déposant ou retirant des messages du tampon partagé.

L'accès au tampon partagé sera géré par une instance de la classe `BufferPartage` assurant le rôle d'un « moniteur » et proposant les opérations permettant de déposer et retirer des messages du buffer de manière synchronisée et donc cohérente.

Le tampon dans lequel s'effectuent les dépôts sera représenté par un tableau (d'entiers, par exemple – ou tout autre type de tableau ou de buffer adéquat) et géré circulairement.

Version 1

Les messages, non typés, sont déposés dans l'ordre FIFO.

Version 2

Les messages sont typés et les dépôts doivent assurer que deux messages consécutifs dans le buffer sont de type différent. On pourra implanter une classe `Message`.

Degré de parallélisme

On désire obtenir des solutions dans lesquelles le degré de parallélisme est le même que dans les solutions implantées avec les sémaphores : un producteur et un consommateur peuvent avoir accès au tampon au même instant sans que cela ne pose problème.

Exercice 2

Programmer une application réalisant un rendez-vous entre `N` threads Java (principe de l'exercice 2 de la feuille 3 de TP), `N` pourra être variable d'une exécution à l'autre.

Une classe `Participant` définira les threads devant se synchroniser à ce rendez-vous.

La synchronisation sera gérée par une classe `Barriere` dont une instance jouera le rôle d'un « moniteur » proposant les opérations permettant aux threads `Participant` de réaliser la synchronisation nécessaire.