Thème 8 - TME

Exercice

Lors de chaque TME, vous devrez créer un répertoire pour chaque exercice, y mettre les fichiers s'y rapportant et soumettre ce répertoire à la fin du TME.

Exercice 11 – De l'utilité d'un serveur multi-threads

Dans un système client/serveur, un ensemble de processus, appelés clients, envoient des requêtes à un processus serveur qui les traite et renvoie à chaque client la réponse correspondant à sa requête. Nous voulons programmer un système de ce type dans lequel :

- le serveur traite une seule requête à la fois : le traitement de chaque requête est exécuté directement dans le thread Serveur.
- un client ne peut envoyer une nouvelle requête qu'après avoir reçu la réponse à sa requête précédente.

Chaque client envoie au serveur une suite de requêtes numérotées. La réponse envoyée par le serveur lorsqu'il a traité une requête inclut une référence au client émetteur (certes inutile pour l'instant), le numéro de la requête et une valeur entière aléatoire représentant le résultat attendu par le client. Chaque réponse du serveur est une instance d'une classe `ReponseRequete`.

Question 1

Proposez une définition pour la classe `ReponseRequete`. Cette définition devra inclure une méthode `toString`.

Question 2

Programmez le thread principal qui crée un thread serveur et `NB_CLIENTS` threads clients. Comment faire pour terminer proprement l'application lorsque toutes les requêtes de tous les clients ont été servies ?

Un client émet vers le serveur des requêtes qui contiennent une référence vers le client émetteur (lui-même), un numéro de requête et un entier permettant de déterminer le type de traitement demandé. Nous considérons ici deux types de requêtes : les requêtes de type 1 émises par des clients dont l'identifiant n'est pas un multiple de 3, et les requêtes de type 2 émises par les autres clients. La soumission d'une requête est effectuée en appelant la méthode `soumettre` de la classe `Serveur`.

Un client qui a émis une requête attend de recevoir la réponse du serveur, effectue localement des opérations (dont l'exécution est représentée par une durée aléatoire) avant de soumettre une nouvelle requête.

Question 3

Donnez le code de la classe `Client` lorsque chaque client envoie successivement 5 requêtes au serveur. Le serveur transmet au client la réponse à une requête en appelant la méthode **`void`** `requeteServie(ReponseRequete r)` de la classe `Client`. Vous devrez vous assurer qu'un client ne se termine pas avant d'avoir reçu une réponse à toutes ses requêtes.

Nous nous intéressons maintenant aux opérations du serveur. Celui-ci étant exécuté par un thread séparé, il doit comporter une méthode `run` dont le code est donné ci-dessous :

```
public void run() {
   try {
      while (true) {
         attendreRequete();
         traiterRequete();
      }
```

```
      }
      catch (InterruptedException e) {
         System.out.println("Serveur interrompu");
      }
   }
```

Question 4

Donnez le code des méthodes `soumettre` (appelée par un client qui dépose une requête) et `attendreRequete` de la classe `Serveur`. Le traitement de la requête étant exécuté dans le thread `Serveur`, la méthode `soumettre` appelée par le client signale simplement au serveur l'arrivée d'une nouvelle requête.

Vous devrez réfléchir aux conditions dans lesquelles un client ou le serveur se trouve bloqué en attente d'une action d'un autre processus.

La durée des opérations de traitement d'une requête de type 1 est représentée par un délai aléatoire borné, alors que nous représentons le traitement des requêtes de type 2 par une boucle infinie (pour simuler le cas d'un traitement très long).

Question 5

Complétez le code de la classe `Serveur` en ajoutant les variables d'instance nécessaires, le code du constructeur et le code de la méthode `traiterRequete`.

Testez l'ensemble du système de tâches. Celui-ci se termine-t-il correctement ?

Pour résoudre le problème de famine créé par l'exécution sur le serveur d'une requête très longue, nous modifions le fonctionnement de celui-ci : lorsqu'il reçoit une requête d'un client, le serveur crée un nouveau thread dédié à l'exécution du traitement de la tâche demandée.

Question 6

Quelles sont les modifications à apporter au système programmé précédemment ? Le problème de famine existe-t-il encore ?