Multithreading

Giuseppe Lipari

13 janvier 2017

1 Compteur parallel

Objectif : Compter le nombre d'occurrences des mots dans un texte en utilisant plusieurs threads (en Java).

Il faut concevoir un programma Java que prend en entrée un fichier texte et calcule les mots les plus fréquemment utilisés dans le texte.

1. Écrire le programme de manière séquentielle. Le programme lit le fichier ligne par ligne. Pour découper une ligne en mots, on peut utiliser la méthode `split()` de la classe `String`, comme dans l'exemple suivant :

```
String txt = new String("this is a test");
String[] result = txt.split("\\s");
for (int x=0; x<result.length; x++)
  System.out.println(result[x]);
```

   Les mots sont mémorisés dans un `HashMap<String, int>` qui associe à chaque mot le nombre d'occurrences trouvées. En fin, le programme imprime le mot avec le plus grand nombre d'occurrences.

   Voici les définition pour la classe HashMap :

```
HashMap<String, Integer> map = new HashMap<String, Integer>();
map.containsKey("et");           // retourne un boolean
map.put("et", new Integer(2));
Integer i = map.get("et");       // il peut retourner null
Map.Entry<String, Integer> entry; // element contenu dans la map
// iteration sur les elements de la map
for (Map.Entry<String, Integer> entry : map.entrySet()) {
   String k  = entry.getKey();
   Integer i = entry.getValue();
}
```

1. Écrire le programme de manière multithreadé. Le programme prend en entré sur la ligne de commande le nombre de threads à créer et le nom du fichier. Les lignes du fichier sont partitionnées entre les threads, et chaque thread compte les mots sur les lignes affectées à lui.

   Question : est-ce que on peut utiliser un seul `HashMap` pour tous les thread ?

2 Mailbox simple

Un système multithread est composé par une `class Mailbox`, par des threads producteurs et par des threads consommateurs.

Les producteurs déposent des données de type `class Data` dans la mailbox, et les consommateurs les prélèvent plus tard. La mailbox peut conserver au maximum N données.

Quand la mailbox est plein, le producteur qui essaie de déposer un nouveau élément est bloqué jusqu'à quand une position devient libre. De la même façon, quand la mailbox est vide, un consommateur qui essaie de prélever un élément est bloqué jusqu'à quand un nouveau élément est disponible.

Coder le système en Java.

3 Mailbox avec controleur

Dans un système multi-thread producteur/consommateur, la structure de données `class Mailbox` contient N messages de type `class Data`. Le système contient 3 thread : le producteur, le consommateur, et un contrôleur.

Quatre opérations sont prévu sur la mailbox :

```
public void produce(Data d);
public void comsume(Data d);
public void stop();
public void start();
```

La première opération est utilisée par le producteur pour insérer un message dans la mailbox ; la deuxième opération est utilisée par le consommateur pour prélever un message.

Le troisième thread (contrôleur) peut invoquer les opérations `start` et `stop`.

— L'opération `stop()` suspends les opérations d'insertion dans la mailbox. Après l'opération de `stop()`, le producteur est bloqués quand il appellent `produce()`. Le consommateur peut continuer à extraire des message (s'il n'y a).

— L'opération de `start()` rétablie le comportement normal : si le producteur est bloqué, il est réveillé et il peut continuer à insérer des messages. Mais, avant de réveiller le producteur, il faut attendre que la mailbox soit vide. Si la mailbox n'est pas vide, le contrôleur est bloqué, et on attend que le consommateur consommes tous les messages.

Il faut coder la class `Mailbox` et les 4 opération décrites.

4 Workers avec mailbox

Dans l'exercice du compteur parallèle, le `main` lit tout le fichier en mémoire avant de lancer les threads. On voudrais que le main lit et envoie les lignes au threads à fur et à mesure. Pour faire ça, on utilise le `Mailbox` développé à l'exercice précédente.

1. Étendre la Mailbox simple pour contenir des `String` (au lieu des entiers).
2. Reprendre le code du compteur parallèle. Créer un Mailbox pour chaque thread. Chaque thread récupère la ligne à découper de son Mailbox au lieu de l'`ArrayList`. Le main met les lignes dans les mailbox.
3. Concevoir une manière pour arrêter les threads quand il n'y a plus de lignes à analyser.