Université Denis Diderot Paris 7
UFR d'informatique
M2-programmation répartie

Examen d'~~algorithmique~~ programmation répartie
(23 mars 2015)

(durée 2 heures - documents non autorisés)
2 pages

Exercice 1.— On considère des objets *séquentiels* ayant un état donné par la valeur d'une variable *val* (dont l'état initial est 0) et une méthode *ajouter(n)* telle que :

si $val = x$ alors $ajouter(y)$ retourne $x + y$ et met la valeur de val à $x + y$.

On appellera cette classe d'objets séquentiels *FAj*.

On considère d'autre part le code A java suivant :

```
class Adder{
 private int val=0;
 public int ajouter(int i){
   int tmp = val;
   val = tmp+i;
   return tmp; val
  }
}
```

1. Ecrire un code java qui crée deux threads partageant un objet `Adder`. (Dans la suite, la première thread sera désignée comme la thread T et la deuxième comme la thread U). Chacune des threads appellera la méthode `ajouter` et affichera la valeur retournée. La thread T appellera `ajouter(1)` et la thread U appellera `ajouter(2)` suivi de `ajouter(1)`.
   - Quelles sont les valeurs possibles affichées par les deux threads ? Quelles sont la valeurs possibles de `val` quand les deux threads ont terminé ? affichage de la val de retour de ajouter(x)
   - Soit d_T (respectivement d_U) l'instant du début l'appel de `ajouter` pour la thread T (respectivement U) et soit f_T (respectivement f_U) l'instant du retour de l'appel de `ajouter` pour la thread T (respectivement U). Décrire (par un diagramme de temps) les postions relatives possibles de de d_T, d_U, f_T, f_U les valeurs retournées et la valeur de val.
   - Rappelez quelles sont les conditions pour qu'un code implémente une spécification séquentielle. Le code A implémente-t-il la spécification séquentielle de *FAj* ?
2. Si la variable `val` de A est déclarée *volatile*, le code A modifié implémente-t-il la spécification séquentielle de *FAj* ?
3. Modifier le code A en déclarant la méthode `ajouter` comme étant `synchronized`, le code modifié implémente-t-il la spécification la spécification séquentielle de *FAj* (justifier la réponse) ?
   Si oui l'implémentation obtenue est-elle « wait free » ?
4. On suppose que l'on dispose d'une classe `Lock` ayant une méthode `lock()` et une méthode `unlock()` qui assure les propriétés d'exclusion mutuelle. Modifier le code A pour obtenir une implémentation de la spécification séquentielle de *FAj* (justifier la réponse) ? Cette implémentation est-elle wait-free ?
5. Montrer qu'avec un objet satisfaisant l'implementation séquentielle de *FAj*, il est possible de réaliser un consensus entre 2 threads. Pour cela on demande d'écrire explicitement le code réalisant ce consensus entre deux threads. (On pourra utiliser le fait que dans le (cas d'une implémentation séquentielle de *FAj*), seul le premier appel de *ajouter* retourne 0).

6. Est-il possible d'avoir une implémentation wait-free d'objets *FAj* uniquement avec des registres atomiques ? (On pourra utiliser le résultat de la question précédente).

Exercice 2.—

On rappelle la spécification séquentielle du *Snapshot*

```
1 public class SeqSnapshot<T> implements Snapshot<T> {
2    T[] a_value;
3    public SeqSnapshot(int capacity, T init) {
4       a_value = (T[]) new Object[capacity];
5       for (int i = 0; i < a_value.length; i++) {
6       a_value[i] = init;
7       }
8    }
9    public synchronized void update(T v) {
10     a_value[ThreadID.get()] = v;
11   }
12   public synchronized T[] scan() {
13     T[] result = (T[]) new Object[a_value.length];
14     for (int i = 0; i < a_value.length; i++)
15     result[i] = a_value[i];
16     return result;
17   }
18 }
```

On considère un ensemble de n threads. Chaque thread i a une identité unique t_i qui est un entier positif et qui est donnée par la variable locale finale `Id` de la thread i.

Ces threads partagent un objet `so` de type `Snaphot<Integer>` qui a été initialisé à -1 (ce qui correspond au code `so=new Snapshot<Integer>(n,-1);`) et qui implémente la spécification séquentielle du snapshot. `Res` est une variable locale et `ArrayToSet(v)` est une méthode qui retourne l'ensemble des entiers différents de -1 dans le tableau `v`.

On suppose que chaque thread exécute le code suivant (une seule fois) :

```
so.update(Id);
Res=ArrayToSet(so.scan());
```

Dans la suite Res_i représente la valeur de la variable `Res` de la thread i. (On prouvera les questions suivantes en utilisant la linéarisabililté de l'implémentations du snapshot)

1. Montrer que pour toute thread i, Res_i contient sa propre identité ($t_i \in Res_i$).
2. Montrer que pour au moins une thread i, Res_i contiendra l'ensemble de toutes les identités ($\{t_1, \cdots t_n\}$).
3. Montrer que pour toutes threads i et j on a $Res_j \subseteq Res_l$ ou $Res_i \subseteq Res_j$.