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

Examen programmation répartie
(26 mars 2018)
(durée 2 heures - documents non autorisés)

Exercice 1.— On considère des objets *séquentiels* dont l'état est la valeur d'une variable *val* (qui peut être intialisée) et ayant une méthode unique *ajouter(n)* telle que :

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

On appellera *SpAd* cette spécification séquentielle. Ecrire une interface Java correspondant à cette spécification. On appellera cette interface `Add`.

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

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

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)`.
   - Quelles sont les valeurs possibles affichées par les deux threads? Quelles sont la valeurs possibles de `val` quand les deux threads ont terminé?
   - Soit d_T (respectivement d_U) l'instant du début de 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 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 *SpAd*?

2. Si la variable `val` de A est déclarée *volatile*, le code A modifié implémente-t-il la spécification séquentielle de *SpAd* ?
3. Modifier le code A en déclarant la méthode `ajouter` comme étant `synchronized`, le code modifié implémente-t-il la spécification séquentielle de *SpAd* (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 *SpAd* (justifier la réponse) ?

Exercice 2.— On considère l'interface :

```
interface Snapshot {
    int[] scan();
    void update(int v, int i);
}
```

Avec la spécification séquentielle :
- Un objet de la classe **Snapshot** a deux opérations **scan** et **update**, l'état est un tableau **Tab** de n entiers (n est le nombre de threads qui partageront cet objet)
- **update(v,i)** modifie le tableau **Tab** en modifiant **Tab** : **Tab[i]:=v**
- **scan** retourne **Tab**.

On considère la classe suivante :

```
class Snap implements Snapshot {
    int[] a_value;
    public Snap(int capacity, int init) {
        a_value =  new int[capacity];
        for (int i = 0; i < a_value.length; i++) {
            a_value[i] = init;
        }
    }
    public synchronized void update(int v, int id) {
        a_value[id] = v;
    }
    public synchronized int[] scan() {
        int[] result =  new int[a_value.length];
        for (int i = 0; i < a_value.length; i++) {
            result[i] = a_value[i];
        }
        return result;
    }
}
```

1. Snap est-elle une implémentation linéarisable de la spécification séquentielle de Snapshot. (Si oui, on justifiera la réponse et si non on donnera un contre-exemple.)
2. Si dans Snap, on supprime les synchronized, obtient-on une implémentation linéarisable de Snapshot. (Si oui, on justifiera la réponse et si non on donnera un contre-exemple.)
3. On considère la classe :

```
class DCollect implements Snapshot {
    int[] a_value;
    public DCollect(int capacity, int init) {
        a_value =  new int[capacity];
        for (int i = 0; i < a_value.length; i++) {
            a_value[i] = init;
        }
    }
    public synchronized void update(int v, int id) {
        a_value[id] = v;
    }
    public int[] collect() {
        int[] result =  new int[a_value.length];
        for (int i = 0; i < a_value.length; i++) {
            result[i] = a_value[i];
        }
        return result;
    }
    public boolean eq(int[] t1, int[] t2) {
        for (int i = 0; i < t1.length; i++)
            if (t1[i] != t2[i]) return false;
        }
        return true;
    }
    public void copy(int[] t1, int[] t2) {
        for (int i = 0; i < t1.length; i++) {
            t1[i] = t2[i];
        }
    }
    public int[] scan() {
        int[] ancien = collect();
        int[] nouveau = collect();
        while (true) {
            if (eq(ancien, nouveau)) {
                return nouveau;
            } else {
```

```
                copy(ancien, nouveau);
                nouveau = collect();
            }
        }
    }
}
```

Un `update` temine-t-il toujours ? Un `scan` termine-t-il toujours ? Que peut-il se passer s'il y a un nombre infini de `update` ? Dans le cas où un `scan` termine, quelle est en fonction des `update` réalisés précédemment la valeur du tableau retourné ?

En supposant que chaque thread ne fait qu'un nombre fini de `update`, `DCollect` est-elle une implémentation linéarisable de la spécification séquentielle de `Snapshot` ?

4. On considère un ensemble de n threads. Chaque thread i a une identité unique t_i qui est un entier positif entre 0 et $n-1$ et que l'on supposera être donnée par la valeur d'une variable locale finale `Id` de la thread i.

   Ces threads partagent un objet `so` de type `Snap` qui a été initialisé à -1 (ce qui correspond au code `so=new Snap(n,-1);`). `Res` est une variable locale et `ArrayToSet(t)` est une méthode qui retourne l'ensemble des entiers différents de -1 dans un tableau `t`.

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

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

   Dans la suite Res_i représente la valeur de la variable `Res` du thread i après ce code.

   (a) Montrer que pour toute thread i, Res_i contient sa propre identité ($t_i \in Res_i$).

   (b) Si tous les threads exécutent leur code jusqu'au bout alors pour au moins un thread i, Res_i contiendra l'ensemble de toutes les identités ($\{t_0, \cdots t_{n-1}\}$).

   (c) Montrer que pour tous les threads i et j on a $Res_j \subseteq Res_i$ ou $Res_i \subseteq Res_j$.

   (d) Si au lieu d'utiliser la classe `Snap`, on utilise la classe `DCollect` aurait-on les même réponses pour les questions (a) (b) et (c) ?