Université Paris 7
M2 II
Programmation Répartie

Exam 17h30-19h

Tous documents écrits autorisés

Exercice 1.— On considère le problème des lecteurs/rédacteurs.

1. Si un lecteur utilise la base de données, et qu'un autre lecteur veut aussi y accéder que doit-il se passer?
2. 2 rédacteurs peuvent-il accéder concurremment aux données?
3. Un lecteur et un rédacteur peuvent-ils accéder concurremment aux données?
4. En utilisant des sémaphores, donner une solution pour ce problème avec une priorité égale pour les lecteurs et les rédacteurs (si un rédacteur attend pour utiliser la base de données plus aucun lecteur ne pourra y entrer).

Exercice 2.— Des threads disposent d'une mémoire partagée. La thread i peut lire la mémoire(**scan**) et écrire dans un l'élément i de celle-ci (**update(x)** écrit x à l'indice i). L'interface est la suivante

```
public interface Snapshot<T> {
    public void update(T v);
    public T[] scan();
}
```

On réalise une première implémentation de **scan** et de **update**:

```
import java.util.concurrent.atomic.AtomicStampedReference;
public class SimpleSnap<T> implements Snapshot<T> {
    private AtomicReference<T> [] a_table;
public SimpleSnap(int capacity, T init){
    a_table=(AtomicReference<T> [])new AtomicReference [capacity];
    for (int i=0;i<capacity;i++)a_table[i]=new AtomicReference<T>(init);
}

    public void update(T v) {
        int me=ThreadID.get();
        AtomicReference<T> nouv=new  AtomicReference<T>(v);
        a_table[me]=nouv;
    }
```

```
public T[] scan(){
    T[] result=(T[]) new Object[a_table.length];
    for(int j=0;j<a_table.length;j++) result[j]=a_table[j].get();
    return result;
    }
}
```

---

1. Donner un exemple d'exécution montrant que dans cette implémentation, `Snapshot<T>` n'est pas atomique.

On ajoute `synchronized` devant chaque méthode de la classe `SimpleSnap<T>`.

2. Est-ce que dans cette implémentation `Snapshot<T>` est atomique?

3. Est-ce que cette implémentation est sans-attente (wait free)?

Exercice 3.— Dans cet exercice tous les objets sont atomiques et sans attente (wait-free). On considère l'objet `SymetryBreaking`. Cet objet n'a qu'une seule méthode $sb()$ qui retourne 0 ou 1. Sa spécification séquentielle est : le premier appel de $sb()$ retourne 0 les autres retournent 1.

1. En utilisant des registres atomiques et des objets `SymetryBreaking` réaliser du consensus pour 2 processus. (On rappelle que le consensus est un objet avec une méthode $propose(int\ v)$ qui retourne un entier et dont la spécification est : tous les *propose* retournent le paramètre du premier *propose*)

2. En utilisant des registres atomiques et des objets `SymetryBreaking`, est-il possible de réaliser du consensus pour 3 processus?

3. Est-il possible de réaliser un `SymetryBreaking` à l'aide de registres atomiques et de `CompareAndSet`. (On rappelle que cet objet n'a qu'une méthode $cas(int\ v, int\ w)$ qui retourne un booléen et dont la spécification est si la valeur de l'objet est égale à v alors elle devient w et la méthode retourne *true* sinon elle retourne $false$ et l'objet ne change pas de valeur.)

4. Est-il possible de réaliser un `CompareAndSet` à l'aide de registres atomiques et de `SymetryBreaking`.