Université Paris Cité

Master 2 - Année 2023-24

Programmation répartie

TP2-implémentation

Le but de ce Tp est d'implémenter une liste chainée représentant un ensemble avec un grain fin de synchronisation
Pour simplifier on supposera que les éléments sont des entiers (Integer). L'interface Ensemble est alors la suivante:

```
public interface Set{
    boolean add( Integer x);
    boolean remove (Integer x);
    boolean contains( Integer x);
}
```

Exercice 1.— On rappelle l'implémentation a gros grain. Elle utilise la classe Node ci dessous. Pour simplifier on supposera que la clef (key) est l'entier (int) correspondant à l'Integer item.

```
public class Node {
 public Integer item;
 public int key;
 public Node next;
 public Node(Integer i){
     item=i;
     key=i;
 }
}


public class MonSet {
    private Node head;
    private Lock lock= new ReentrantLock();
    public MonSet(){
        head= new Node(Integer.MIN_VALUE);
        head.next= new Node(Integer.MAX_VALUE);
    }
    public boolean add(Integer item){
        Node pred, curr;
        int key= item;
        lock.lock();
        try{
            pred=head;
            curr=pred.next;
            while (curr.key<key){
                pred=curr;
                curr=curr.next;
            }
            if ( key==curr.key) {return false;}
            else {Node node= new Node( item);
```

```
        node.next=curr;
        pred.next=node;
        return true;
        }
    } finally {lock.unlock();
       }
    }

public boolean contains(Integer item){
    Node pred, curr;
    int key= item;
    lock.lock();
    try{
        pred=head;
        curr=pred.next;
        while (curr.key<key){
            pred=curr;
            curr=curr.next;
        }
        if (key==curr.key) {return true;}
        else {
        return false;
        }
    } finally {lock.unlock();
        }
    }

public boolean remove(Integer item){
    Node pred, curr;
    int key= item;
    lock.lock();
    try{
        pred=head;
        curr=pred.next;
        while (curr.key<key){
            pred=curr;
            curr=curr.next;
        }
        if (key==curr.key) {
            pred.next=curr.next;
            return true;}
        else {
        return false;
        }
    } finally {lock.unlock();
        }
    }
```

1. Cette implementation est-elle linéarisable (si oui donnez les points de linéarisation et justifiez sinon donnez un exemple)

2. Est-elle wait-free?

3. Ecrire une thread qui utilise cette implémentation (le constructeur de la thread aura en paramètre un Set) Si cette thread a pour identité i (donne par ThreadID) elle ajoute i, $2 * i$, $i + 4$, $i + 3$ à l'ensemble, teste si $i + 5$ y appartient et enlève $i + 4$

4. Ecrire un programme qui lance 3 threads qui partagent un Set.

Exercice 2.— On veut implémenter cette interface en utilisant une liste chainée et une synchronisation à grain fin.
On utilise pour cela la classe Node

```
class Node {
           Integer item;
           int key;
           Node next;
           ReentrantLock lock;
 }
```

On supposera toujours que la clef (key) est l'entier (int) correspondant à l'Integer item.

1. Ecrire une implementation de Set en utilisant une synchronisation à grains fins

2. Cette implementation est-elle linéarisable (si oui donnez les points de linéarisation et justifiez sinon donnez un exemple)

3. Est-elle wait-free?

4. Réutilisé le programme de l'exercice précédent qui lancent 3 threads et les threads qui ajoutent et enlèvent des éléments avec cette implémentation.