TP

Exclusion mutuelle et Verrou

Dans le package `java.util.concurrent.locks` se trouve l'interface `Lock`.

```
public interface Lock{
    public void lock();
    public void unlock();
}
```

Le but de ce TP est d'implémenter l'interface `Lock` par l'algorithme de Peterson pour 2 threads, puis la généralisation à n threads. Ecrire le code d'une thread qui rentre régulièrement en section critique en utilisant un verrou (la section critique et la section non critique pourront être simulé par une mise en sommeil de la thread).

1 Exclusion mutuelle pour 2 threads

On rappelle le code de l'algorithme de Peterson.

```
private volatile boolean[] flag= new boolean[2];
private volatile int victim;

public Peterson(){
    for(int i=0 ; i<flag.length ; ++i)
        flag[i] = false;
}

public void lock() {
    int i = ThreadID.get() % 2;
    int j = 1 - i;
    flag [i]=true; // I'm interested
    victim = i ; // you go first
    while ( flag[j] && victim == i) {}; // wait
}

public void unlock() {
    int i = ThreadID.get() % 2;
    flag[i]=false; // I'm not interested
}
```

Ecrire l'implémentation de `Lock` par cet algorithme.

2 Exclusion mutuelle pour n threads

Pour implementer le verrou on propose d'utiliser l'algorithme de Peterson généralisé à n threads dont on rappelle le code ci dessous:

```
class Filter implements Lock {
static final int IDLE = -1;
    volatile int[] level;
    volatile int[] victim;
    volatile int size;
    public Filter(int threads) {
        size = threads;
        level = new int[threads];
        victim = new int[threads - 1];
    }

    public void lock() {
        int me = ThreadID.get();
        for (int i = 0; i < size - 1; i++) {
            level[me] = i;
            victim[i] = me;
            // spin while conflicts exist
            while (sameOrHigher(me, i) && victim[i] == me) {};
        }
        level[me] = size - 1;
    }
    // Is there another thread at the same or higher level?
    private boolean sameOrHigher(int me, int myLevel) {
        for (int id = 0; id < size; id++)
            if (id != me && level[id] >= myLevel) {
                return true;
            }
        return false;
    }
    public void unlock() {
        int me = ThreadID.get();
        level[me] = IDLE;
    }
}
```

Répondre aux questions suivantes (si la réponse est positive donnez un exemple d'exécution (ou indiquez quelle exécution donnerait ce résultat) si la réponse est négative expliquez pourquoi)

1. Peut il y avoir 10 Threads au niveau 0 simultanément?
2. Peut il y avoir 10 Threads au niveau 1 simultanément?

3. Si il y a au moins 1 thread au niveau 0 et aucune au niveau 1 est ce qu'une des threads de niveau 0 peut passer au niveau 1?
4. Si il y a au moins 1 thread au niveau 0 et au moins 1 thread au niveau 1 est ce toutes les threads de niveau 0 peuvent passer au niveau 1?
5. Peut il y avoir 9 Threads au niveau 1 simultanément?
6. Peut il y avoir 9 Threads au niveau 2 simultanément?
7. Peut il y avoir 8 Threads au niveau 2 simultanément?
8. Si une thread accède à la section critique, à quels niveaux sont les autres threads ?
9. Montrer la propriété suivante: pour tout j , $0 \leq j \leq size - 1$, il y a au plus $n - j$ threads au niveau j .
10. Montrer que une thread qui demande l'accès à la section critique parviendra en section critique.