UNIVERSITÉ DE GENÈVE — Département d'Informatique
Faculté des Sciences — Mardi 4 février 2020

Examen
Systèmes concurrents et distribués

Exercice 1: Implémentez une solution au problème du barbier endormi qui utilise les mécanismes de synchronisation Java.
Pour ce problème on a un processus Barbier et plusieurs processus Clients. Dans le salon du barbier il y a n chaises pour attendre. Lorsqu'un client arrive:

- Si les n chaises sont occupées il ressort.
- Si un client se fait raser et qu'au moins une chaise est disponible alors il s'assied et attend.
- S'il n'y a pas de client dans le salon alors il réveille le barbier et s'assied dans la chaise pour se faire raser.

Pour le barbier :

- S'il n'y a pas de clients dans le salon, il s'endort.
- Lorsqu'il a terminé de raser un client il s'occupe du client suivant s'il y en a un, sinon il s'endort.

Pour simuler que le barbier rase un client, chaque client va transmettre un nombre aléatoire au barbier qui l'affiche avec l'identificateur du client.

Exercice 2: Avec le code ci-dessous pour la gestion d'un tampon.

- Montrez qu'avec un producteur (appelle à enq()) et deux consommateurs (appelle à deq()) alors il existe une exécution qui conduit à un interblocage.
- Montrez que si $QSIZE = 2$ alors il existe une exécution qui résulte en un interblocage avec un producteur et 4 consommateurs.

```
1 public class queue {
2 private int head = 0, tail = 0;
3 final int QSIZE=1;
4 Item[QSIZE] items;
5 public synchronized void enq(Item x) {
6         while (this.tail - this.head == QSIZE)
7                 this.wait();
```

```
8           this.items[this.tail++ % QSIZE] = x;
9           this.notify();
10  }
11  public synchronized Item deq(Item x) {
12          while (this.tail - this.head == 0)
13                  this.wait();
14          Item inter = this.items[this.head % QSIZE];
15          this.head+=1;
16          this.notify();
17          return inter;
18  }
19  }
```

Exercice 3: Décrivez l'algorithme de Lamport pour résoudre le problème de l'exclusion mutuelle dans les **systèmes distribués**. En particulier décrivez clairement les conditions qui assurent qu'un processus peut accéder à la ressource partagée.
Montrer que si un processus i satisfait les conditions pour accéder la ressource partagée alors un processus $j \neq i$ ne satisfait pas les conditions. Pour rappel, on suppose que les messages ne sont jamais perdus et qu'ils arrivent dans l'ordre.

Exercice 4: Pour la programmation concurrente en Java le modèle de mémoire décrit les conditions qui assurent qu'une écriture par un thread dans une variable est visible par un autre thread.

- Donnez deux exemples de conditions qui assurent qu'une écriture est visible par un autre thread.
- De manière générale, quelle est la condition qui assure qu'une écriture est visible par un autre thread.
- Pourquoi est-ce que le language Java n'assure pas que toutes les écritures soient visibles?

Exercice 5:

- Donner les trois conditions formelles pour qu'un registre soit atomique. Utilisez les notations W^i pour l'écriture de la valeur v^i et R^i pour sa lecture.
- Décrivez les situations où ces conditions ne sont pas satisfaites.

```
1  class MultiValued {
2  int n = 0;
3  boolean A[] = null;
4  public MultiValued(int maxVal, int initVal) {
5  n = maxVal;
6  A = new boolean [n];
7  for(int i = 0; i < n; i++)
8          A[i] = false;
9  A[initVal] = true;
10 }
11
12 public void setValue(int x){
13 A[x] = true;
14 for( int i = x -1; i >= 0; i--)
15         A[i] = false;
16 }
17
18 public int getValue() {
19 int j = 0;
20 while(!A[j])
21         j++;
22 return v;
23 }
24 }
```

Pour le code de la classe MultiValued.

- Est-ce correct de déplacer l'instruction à la ligne 13 après l'exécution de la boucle **for** aux lignes 14 et 15, i.e. à la ligne 16.
- Montrez que le registre implémenté par la classe MultiValued n'est pas atomique.

Exercice 6: L'algorithme de Kessel pour 2 threads résoud le problème de section critique. Les threads Thread 0 et Thread 1 exécutent les codes ci-dessous.

- Expliquez en quoi consiste le problème de section critique.
- Expliquez deux propriétés que doit satisfaire une solution (excepté l'équité).
- Montrez que la solution de Kessel pour 2 threads assure qu'un seul processus à la fois exécute la section critique.

```
1 Thread 0
2 b[0]=true;
3 local[0]=turn[1];
4 turn[0]=local[0];
5 while ((b[1]==true && (local[0]==turn[1]))
6
7 section critique
8
9 b[0] = false;
```

```
1 Thread 1
2 b[1]=true;
3 local[1]=1-turn[0];
4 turn[1]=local[1];
5 while ((b[0]==true && (local[1]!=turn[0]))
6
7 section critique
8
9 b[1] = false;
```

- Expliquez pourquoi l'algorithme est équitable.

Exercice 7: On considère le problème du consensus pour deux processus qui communiquent par mémoire partagée.

- Expliquez le problème du consensus et ce qu'est une solution wait-free.
- Expliquez ce que sont un état bivalent et un état critique.
- Montrez qu'une solution wait-free au problème du consensus possède toujours un état critique et que l'état initial est toujours bivalent.
- Démontrez qu'il n'existe pas de solution wait-free à ce problème pour deux processus qui communiquent par des écritures/lectures atomiques en mémoire partagée.