DS 2011 Parallélisme

Damien Crémilleux

25 mars 2014

1 Question de cours

- Tout partage de variable impose une synchronisation :
  Vrai, la synchronisation est nécessaire pour lire et écrire de façon cohérente.
- Les compteurs de synchronisation sont un mécanisme de syncronisation de bas niveau :
  Faux, il s'agit d'un mécanisme de haut niveau. En effet, le réveil est implicite, on ne l'écrit pas.
- En Java l'attribut private rend une variable locale à un thread et il faut utiliser shared pour indiquer qu'elle est partagée :
  Faux
- En Java on choisit quel est le thread que l'on réveille avec la méthode notify(Thread) :
  Faux, on ne choisit pas le thread qui sera réveillé.
- En Java Notify réveille un thread qui reprend instantanément le contrôle de l'objet partagé :
  Faux, il faut attendre l'acquisition du verrou (d'où le *while* pour l'attente). C'est donc différent par rapport aux moniteurs de Hoare.
- Les méthodes *synchronized* s'exécutent en exclusion mutuelle avec toutes les autres méthodes du même objet :
  Faux, les méthodes non-*synchronized* ne sont pas impactées. En outre, les *synchronized* sont réentrants (appel à une méthode *synchronized* dans une méthode déjà *synchronized*).
- Un programme parallèle va toujours plus vite qu'un programme séquentiel :
  Faux, à cause des synchronizations un programme parallèle peut être plus long qu'un programme séquentiel.
- Un programme parallèle ne peut s'exécuter que sur une machine parallèle ou multi-coeur :
  Faux, il suffit que la machine supporte le multi-threading.
- La communication par rendez-vous est le moyen de communication le plus rapide dans un programme parallèle exécuté sur un réseau de machines :
  Faux, en effet il faut attendre l'émetteur (principe de l'accusé de réception).
- MPI utilise la communication par rendez-vous
  Faux, MPI utilise la communication avec envoi non-bloquant.
- Les variables globales dans un programme C/MPI sont partagées entre les processus
  Faux, il n'y a pas de variables globales en C/MPI (pas de mémoire partagée non plus).
- MPI permet le partage de variable

Faux.

2 Exercice 1 : Synchronisation

Journal.java

```
// Classe Journal pour le DS2011 de parallelisme

public class Journal {

    private int lireAct;
    private int lireAtt;
    private int ecrireAct;
    private int ecrireAtt;

    public synchronized void debut_lire() {
        while(ecrireAct != 0 || ecrireAtt >= 3) {
            lireAtt++;
            this.wait();
            lireAtt--;
            this.notifyAll(); //place du notify ?
        }
        lireAct++;
    }

    public String lire() {
        //effectue la lecture
        return "";
    }

    public synchronized void fin_lire() {
        lireAct--;
        this.notifyAll();
        return;
    }

    public synchronized void debut_ecrire {
        while(ecrireAct != 0 || lireAct != 0) {
            ecrireAtt++;
            this.wait();
            ecrireAtt--;
            this.notifyAll(); //place du notify ?
        }
        ecrireAct++;
    }

    public void ecrire(String s) {
        //ecriture
        return;
```

```
    }

    public void fin_ecrire() {
        ecrireAct--;
        this.notifyAll();
        return ;
    }
```

3 Exercice 2 : MPI