Ecole Polytechnique

Cours : "Parallélisme"
Travaux dirigés
E. Goubault & S. Putot

TD 2

14 janvier 2009

1 Tri parallèle

On suppose que l'on dispose d'un réseau linéaire de p processeurs P_1, ..., P_p, c'est à dire que les processeurs sont organisés sur une ligne, et que chacun peut communiquer avec son voisin de gauche et son voisin de droite s'il en a un. On veut trier n entiers, avec p qui divise n. Chaque processeur a au départ $\frac{n}{p}$ entiers, et on veut trouver un algorithme parallèle qui permette de faire en sorte que la sous-suite sur P_i soit ordonnée et inférieure à la sous-suite ordonnée P_j, pour tout $i < j$. C'est à dire tout simplement que l'on veut trier en parallèle les données réparties sur les processeurs P_i.

En s'inspirant du réseau de tri pair-impair vu en cours, écrire un tel algorithme en JAVA.

2 Lecteurs/rédacteurs

On considère maintenant une base de données partagée accédée par des threads JAVA de deux types. L'un est le type *lecteur*, l'autre le type *rédacteur*.

Les lecteurs sont des threads qui par définition ne peuvent que lire (interroger) la base de données. Les rédacteurs par contre peuvent lire et écrire. On suppose que la lecture parallèle de la même location dans la base de données est autorisée, par contre l'écriture en parallèle avec soit une lecture soit une autre écriture sur la même location doit être effectuée en section critique (exclusion mutuelle).

On simulera (il ne sera pas nécessaire d'avoir une vraie donnée, une simple simulation des actions à l'écran suffit) le début de l'accès en lecture par la méthode (à écrire) `startRead()`, la fin par `end Read()`, le début de l'accès en écriture par `startWrite()` et la fin par `endWrite()`.

On veut implémenter un tel système avec au moins la propriété suivante: aucun lecteur ne doit rester en attente sauf si un rédacteur a obtenu la permission d'utiliser la base de données. Peut-il y avoir des cas de famine?

On pourra également (optionnel) se poser la contrainte supplémentaire suivante: dès qu'un rédacteur est prêt, il doit effectuer son écriture "aussi vite que possible".

Remarque: on pourra soit utiliser la classe `Semaphore` du cours (combien en faut-il et de quel type alors?), soit directement les mots-clés JAVA `wait()` et `notify()`.