Ecole Polytechnique

**Cours : "Parallélisme"**
**Travaux dirigés**
**E. Goubault & M. Martel**

**TD 4**

4 février 2002

## 1 Points morts

(1) On considère un système constitué d'un 4-sémaphore $x$ auquel accèdent 3 processus PV en parallèle, chacun d'entre eux n'ayant jamais besoin de plus de deux verrous sur $x$ à tout instant. Montrer qu'il ne peut pas y avoir d'interblocage.

(2) On généralise la question 1: on considère maintenant un $m$-sémaphore $x$, auxquels accèdent $n$ processus. On supposera que chacun des processus a besoin à tout instant de 1 à $m$ verrous sur $x$ et que la somme des besoins de chacun des processus est inférieur strictement à $m + n$. Montrer que le système est sans interblocage.

## 2 Ordonnanceur à tourniquet

On cherche à implémenter un ordonnanceur à tourniquet pour les threads JAVA en JAVA.

L'idée est que l'on doit définir un thread ordonnanceur qui gère une file d'attente de threads "utilisateurs" JAVA et les ordonnancer un à un pour un quantum de temps élémentaire donné (une constante de votre programme). Cet ordonnanceur ne gèrera pas de notion de priorité.

Chaque thread utilisateur devra s'enregistrer auprès de l'ordonnanceur en appelant la méthode `addThread()` (à écrire). On pourra aussi écrire une méthode `removeThread` qui désinscrit un thread auprès de cet ordonnanceur. On utilisera la classe `CircularList` donnée sur la page web du cours, qui fournit une implémentation de liste circulaire, avec les méthodes:

- `public synchronized void insert(Object o)` d'insertion d'un objet dans la file.
- `public synchronized void delete(Object o)` qui efface l'objet de la liste.
- `public synchronized Object getNext()` qui prend l'élément suivant dans la file.

Avant de commencer, on répondra aux questions suivantes: quelle doit être la priorité du thread ordonnanceur? du thread utilisateur actif? des threads non encore actifs?

## 3 Barrières de synchronisation

Ecrire (avec `wait` et `notify`/`notifyall`) une classe `Barrier` qui implémente les barrière de synchronisation. Une barrière de synchronisation est un mécanisme qui permet d'attendre que `n` (quelconque, donné à la construction) processus s'attendent à un point donné de leur exécution, avant de tous repartir ensemble. On programmera:

- le constructeur `Barrier(int nThreads)` qui spécifie le nombre de processus à synchroniser

- `waitForRest()` qui bloque le processus courant jusqu'à ce que $n-1$ autres processus arrivent à l'exécution de cette même instruction
- une classe exemple qui utilise de façon simple une barrière de synchronisation

# 4 Programmer l'algorithme de tri de Batcher

On associera un processus JAVA par comparateur.

# 5 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()`.