

Corrigé Série 4

Série rendue le: 23 mai 2005

Exercice 1 : Sémantique des moniteurs Java

1.1 Les états d'un processus

activable Etat d'un processus susceptible d'obtenir le processeur. Dans chaque moniteur, un processus au maximum peut être dans cet état.

stoppé Etat d'un processus qui désire s'exécuter dans un moniteur occupé.

attente Etat d'un processus en attente d'un signal.

sorti Etat d'un processus se trouvant en dehors du moniteur.

1.2 Les actions qui vont entraîner un changement d'état d'un processus

entrée Action qui correspond à l'appel d'une procédure du moniteur.

wait Action qui correspond à l'appel de la méthode `wait`.

notify Action qui correspond à l'appel de la méthode `notify`

notifyAll Action qui correspond à l'appel de la méthode `notifyAll`.

sortie Action qui correspond à la fin de l'exécution d'une procédure du moniteur.

1.3 La liste des transitions

Etat	Action	Etat
(sorti, activable)	entrée	(stoppé, activable)
(sorti)	entrée	(activable)
(activable, stoppé)	wait	(attente, activable)
(activable)	wait	(attente)
(activable, attente)	notify	(activable, stoppé [pour un seul processus en attente])
(activable)	notify	(activable)
(activable, attente)	notifyAll	(activable, stoppé [pour tous les processus en attente])
(activable)	notifyAll	(activable)
(activable, stoppé)	sortie	(sorti, activable)
(activable)	sortie	(sorti)

Exercice 2 : Lecteurs-rédacteurs en Java

Les fichiers source sont téléchargeables depuis la page du cours.

Exercice 3 : Producteur - consommateur

a. Lorsqu'on remplace l'appel à `notifyAll()` par un appel à `notify()` dans les méthodes `déposer()` et `prélever()`, la solution Java vue au cours est toujours correcte avec un seul producteur et un seul consommateur. En effet, lorsque le consommateur s'exécute, seul le producteur peut être en attente, et vice-versa.

b. Pour le cas avec n_p producteurs et n_c consommateurs, où $n_p > 1$ et $n_c > 1$, on peut aboutir à un interblocage. Soit un tampon de taille 1, considérons en effet le scénario suivant :

1. C_1 appelle `prélever()`, obtient le verrou, se bloque sur `wait()` (tampon vide)
2. C_2 appelle `prélever()`, obtient le verrou, se bloque sur `wait()` (tampon vide)
3. P_1 appelle `déposer()` et dépose son message dans le tampon (tampon plein), appelle `notify()` et libère le verrou (et C_1) et sort
4. P_2 appelle `déposer` (pendant l'étape 3, alors que P_1 le possède encore) et se bloque sur le verrou
5. P_2 entre (après la libération du verrou par P_1) et se bloque sur `wait()` (tampon plein)
6. C_1 consomme le message m_1 , appelle `notify()` et réveille C_2 et sort
7. C_2 se bloque (tampon vide)

Le système est maintenant bloqué. Ce scénario est généralisable pour le cas d'un tampon de taille $m \geq 1$.