IUT Blagnac
"Introduction aux systèmes temps réel"

Contrôle écrit

28 mars 2007. Durée 90 minutes.
Documents autorisés. Ordinateurs et téléphones portables interdits.

1 Programmation concurrente

Ex.1.1 (4pts) : On considère une ressource qui est accessible en *lecture* ou en *écriture* (par exemple, un fichier). La ressource doit être protégée contre plusieurs accès parallèles en écriture, ou contre un accès en écriture fait en parallèle avec des accès en lecture. Toutefois, plusieurs accès en lecture peuvent être faits en parallèle.

En se basant sur l'existence d'une classe `Semaphore` avec deux primitives, `P()` et `V()`, donner l'implémentation en Java d'une classe `ReaderWriterManager` qui réalise ce type de protection. La classe doit fournir (au moins) 2 méthodes: `read(Reader r)` et `write(Writer w)`, qui réalisent la protection désirée et font ensuite appel à `r.doRead()` et respectivement `w.doWrite()` pour la lecture/écriture effective.

N.B. L'implémentation doit être équitable, au sens où un appel à `write` ne doit pas être retardé à l'infini à cause d'appels successifs à `read` qui se recouvrent dans le temps. (Une implémentation non équitable sera notée zero.)

Ex.1.2 (2pts) :. Justifiez pourquoi on peut dire que l'algorithme de Peterson est équitable ?

Ex.1.3 (4pts) :

a) Expliquez quelle est la difference entre les primitives `wait` et `notify` de la classe Java `java.lang.Object` et les primitives `wait` et `signal` d'un sémaphore binaire.

b) Dans quel cas les primitives Java peuvent-elles lever une exception `IllegalMonitorStateException`?

2 Ordonnancement temps réel

Ex.2.1 (5pts) : On considère un ensemble de 5 tâches avec les périodes et les temps de calcul suivantes:

tâche	période	temps CPU (ms)
A	10	2
B	120	5
C	15	5
D	75	10
E	20	4

Ces tâches partagent 4 ressources R_1, R_2, R_3 et R_4. Le temps d'exécution de la section critique correspondante à chaque ressource, dans toutes les tâches qui l'utilise, est respectivement de 3ms pour R_1, 1ms pour R_2, et 2ms pour R_3 et R_4. L'utilisation des ressources est donnée dans le tableau 1 (page 2).

Les tâches sont ordonnancées selon une politique préemptive à priorités fixes, avec utilisation du protocole d'héritage de priorités plafond immédiat (PCP immédiat).

tâche	ressources utilisées
A	R_2
B	R_1,R_3
C	R_2,R_3,R_4
D	aucune
E	R_1,R_2

Figure 1: Ressources utilisées par les tâches.

1. Quelle est l'affectation optimale des priorités pour ces tâches? Justifiez.
2. Calculez le temps de blocage maximal que peut subir chaque tâche à chaque activation.
3. Effectuez une analyse de temps de réponse pour chaque tâche (de la plus prioritaire à la moins prioritaire) et déterminez si le système est ordonnanceable.

Ex.2.2 (2pts) : Soit un système composé de 2 tâches périodiques: A de période 5ms et temps CPU 2ms, et B de période 20ms et temps CPU 12ms. Quelle est l'utilisation globale du CPU (en pourcentage)? Est-ce que le système est ordonnanceable avec une politique à priorités fixes? Justifiez.

Ex.2.3 (3pts) : Soit 4 tâches A,B,C,D, ayant des priorités telles que $A > B > C > D$. Les tâches partagent des ressources protégées par deux sémaphores binaires a et b. La Figure 2 montre l'exécution de ces 4 tâches, avec les moments des appels aux primitives P et V de a et b, si aucun protocole d'héritage de priorités n'est utilisé.

Modifier ce diagramme pour montrer l'exécution dans le cas où l'on utilise l'héritage de priorités plafond immédiat (PCP immédiat). (On suppose les mêmes durées relatives d'exécution pour chaque tâche, en particulier entre chaque 2 appels consécutifs vers des primitives sémaphore.)

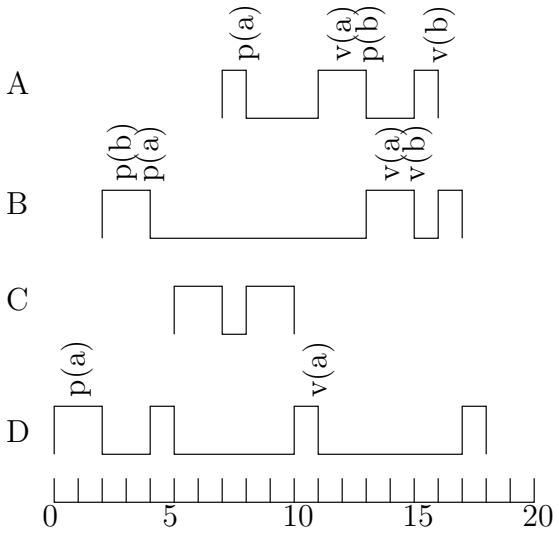

Figure 2: Comportement des 4 tâches.