TD récapitulatif

Systèmes Temps Réel

March 21, 2007

1 Ordonnancement

1. Soit un système formé des 3 tâches avec les caractéristiques suivantes:

tâche	période	temps CPU (ms)
P	100	30
Q	5	1
S	25	5

Intégrer une tâche sporadique R avec temps minimum entre arrivées 50ms, temps CPU 2ms, et échéance relative 4ms.

Quel niveau de priorité faut-il donner à R. Est-ce que le système reste ordonnanceable dans l'ensemble?

2. 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 1 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'héritage simple de priorités (PIP) est utilisé. (On suppose les mêmes durées relatives d'exécution pour chaque tâche, en particulier entre chaque entre 2 appels consécutifs à des primitives sémaphore.)

2 Programmation : synchronisation et temporisation

3. 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

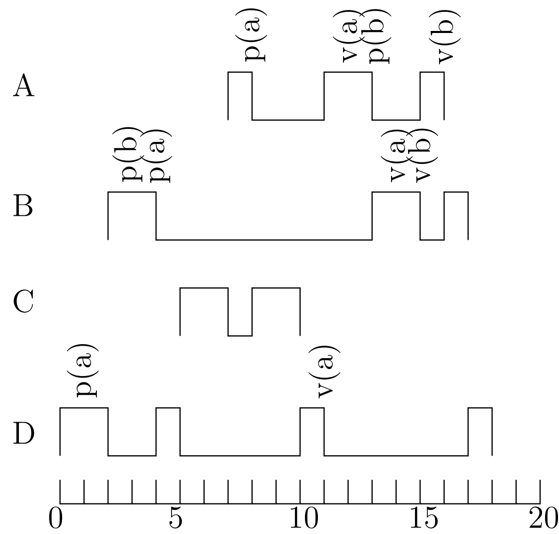

Figure 1: Comportement des 4 tâches.

accès en lecture. Toutefois, plusieurs accès en lecture peuvent être faits en parallèle.

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)`
`write(Writer w)`

qui réalisent la protection et qui font appel à `r.doRead()` et respectivement `w.doWrite()` pour la lecture/écriture effective.

L'implémentation peut se baser sur l'existence d'une classe `Semaphore` avec deux primitives, `P()` et `V()`.

Analyser si l'implémentation que vous avez donné peut "affamer" (i.e., ne jamais donner le CPU à) un thread qui appelle `write` dans le cas ou les appels successifs à `read` se recouvrent. Comment peut-on rendre la situation plus équitable?

4. Donner l'implémentation en Java d'une classe `Sleeper` fournissant 2 méthodes *statiques* `tick` et `sleep(int n)`. La méthode `tick` est appelée par un service d'horloge externe, pour signifier le passage d'une unité de temps. La méthode `sleep` peut être appelée par un thread pour se mettre en suspension pendant `n` "ticks".

L'implémentation ne doit utiliser que les primitives de synchronisation de base du langage Java.