TD 3 - ASR7 Programmation Concurrente

—

Ordonnancement

Yves Caniou, Matthieu Moy, Frédéric Suter

Automne 2018

I Ordonancement avec des mutexes

Nous utilisons un ordonnancement préemptif avec priorité [1]. Nous allons utiliser un jeu de tâches qui mélange des tâches périodiques et des tâches ponctuelles. De plus, deux tâches partagent un mutex.

Tâche	Date(s) d'arrivée(s)	Priorité	Durée	Remarque
A	0, 6, 12, 18, 24, 30	10	1	Périodique
B	0, 10, 20, 30	8	4	Périodique, à chaque itération, la tâche doit acquérir le mutex à la fin du temps 1 et la libérer à la fin du temps 3.
C	18	5	6	Ponctuelle
D	0	1	8	Ponctuelle, à la fin du temps 6, la tâche acquiert le mutex et le conserve jusqu'à la fin du temps 8.

Q.I.1) - Faire l'ordonnancement de ces tâches sur 32 unités de temps.

Q.I.2) - Quel est le temps de réponse de chaque tâche ?

Q.I.3) - Que remarque-t-on aux temps 21 à 27 ?

II Ordonancement

Nous allons utiliser l'implantation POSIX 1003b sur Linux. Il existe 100 niveaux de priorité :
- Le niveau 0 est réservé à SCHED_OTHER et les niveaux de priorité 1 à 99 aux politiques SCHED_FIFO et SCHED_RR.
- Les tâches de priorité 99 sont les tâches de plus forte priorité.
- SCHED_OTHER est dédié à l'ordonnanceur temps partagé.
- Le quantum utilisé par la politique SCHED_RR est de 2 unités de temps.
- Pour SCHED_FIFO et SCHED_RR, lorsqu'une nouvelle tâche arrive, elle est placée en queue (i.e. les tâches sont exécutées dans l'ordre d'arrivée).
- L'ordonnancement est préemptif.

Soit le jeu de tâches apériodiques suivant :

1. Plus la valeur de priorité est importante plus la tâche est prioritaire

Tâche	Date d'arrivée	Priorité	Durée	Politique
o1	0	0	5	SCHED_OTHER
rr1	7	5	2	SCHED_RR
rr2	10	10	6	SCHED_RR
rr3	6	10	7	SCHED_RR
fifo1	1	10	4	SCHED_FIFO
fifo2	3	10	2	SCHED_FIFO

Q.II.1) - On suppose qu'une fois arrivées, les tâches sont toujours prêtes. Dessinez de l'instant 0 à l'instant 26, l'ordonnancement généré par l'ordonnanceur.

Q.II.2) - Donner le temps réponse de chaque tâche.

III Le pont de Miralonde

Le pont de Miralonde est trop étroit pour que 2 voitures puissent se croiser. Vous devez mettre en place un système qui évitera tout incident. Le système se déclenchera automatiquement à l'arrivée d'une voiture à l'une des extrémités du pont. Il autorisera ou non le passage en fonction de la configuration sachant que :

- Si le pont est vide, la première voiture qui arrive peut passer.
- Si le pont contient une voiture qui va du nord au sud, seules les voitures circulant dans le même sens (donc arrivant au nord) peuvent passer.
- Inversement, si le pont contient une voiture qui va du sud au nord, seules les voitures arrivant au sud ont l'autorisation de passer.

Pour éviter tout problème, votre système dispose de barrières contrôlées par un ordinateur. Vous devez écrire le programme de contrôle qui tourne sur cet ordinateur en utilisant les outils classiques (mutex, variables de conditions, ...). À chaque fois qu'une voiture arrive à l'extrémité nord du pont, le système appelle automatiquement la fonction `EntreeNS()` puis lorsque cette voiture sort du pont, la fonction `SortieNS()` est appelée. Inversement, les fonctions `EntreeSN()` et `SortieSN()` servent à gérer les voitures qui circulent dans l'autre sens. Toutes ces fonctions peuvent être appelées en concurrence.

Nous supposerons que si la fonction `EntreeNS()` (ou `EntreeSN()`) se termine, le véhicule est autorisé à passer, alors que si elle bloque, le véhicule est aussi bloqué (par exemple, on peut imaginer un système qui détecte l'arrivée d'une voiture et appelle `EntreeNS()` quand la voiture arrive, lève la barrière d'entrée quand la fonction termine, et la redescend immédiatement après le passage de la voiture).

Q.III.1) - Donnez un algorithme des fonctions `EntreeNS()`, `EntreeSN()`, `SortieNS()` et `SortieSN()` en utilisant le principe du moniteur de Hoare.

Q.III.2) - Montrez la propriété de sûreté : il n'y a jamais à la fois une voiture venant du nord et une venant du sud sur le pont.

Q.III.3) - Montrez que cet algorithme n'a pas de problème de *deadlock* (inter-blocage).

Q.III.4) - L'algorithme pose-t-il un problème de famine ? Si oui, donnez un exemple puis proposez un nouvel algorithme.