TD 2 - ASR7 Programmation Concurrente

—

Thread et concurrence

Matthieu Moy, Fabien Rico, Adil Khalfa

Printemps 2018

I Retour sur le TD1

— Exercices non-faits pendant le TD1

II Pourquoi faire un programme multithread

On souhaite comparer l'efficacité d'un serveur de fichiers en mode mono ou multithread, même sur un ordinateur disposant d'un seul processeur monocoeur.

L'intérêt du multithread se trouve uniquement lors des accès disque car le thread qui demande l'accès à un fichier doit attendre pendant que les données sont lues sur le disque. Le serveur de fichiers dispose d'un cache en mémoire pour les fichiers les plus couramment lus. On suppose :

— la durée pour traiter une requête sans accès disque est de 15ms (récupérer la requête, chercher dans le cache, rendre le résultat) ;
— pour gérer le multithreading un surcoût de 5ms est nécessaire (changement de contexte et passage en mode noyau pour passer d'un thread à l'autre) ;
— si le fichier ne se trouve pas en cache il faut 75ms supplémentaires pour la lecture sur le disque ;
— en moyenne un fichier est disponible dans le cache dans 2/3 des cas.

Q.II.1) - Donner le nombre de requêtes traitées par seconde pour un serveur monothread

Q.II.2) - Donner le nombre de requêtes traitées par seconde pour un serveur multithread utilisant des threads noyaux.

Q.II.3) - Est-il intéressant d'utiliser des threads utilisateurs (green threads) ?

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.