TD 1 - LIF12 système d'exploitation

Passage d'informations sur les pipes

22 mars 2017

I Algorithme de Dijkstra

L'article donnant le premier algorithme de résolution du mutex est donné page **??**. Il date de 1965 et a été conçu avant les sémaphores. Il suppose n processus sur N processeurs, et s'exécute en **mémoire partagée** : Il suppose que K est accessible en lecture par tous et peut être mis à jour par tous.

Q.I.1) - Que sont I, K ?

Q.I.2) - Comment sont initialisés les tableaux B et C ?

Q.I.3) - Montrer que deux processus ne peuvent entrer en section critique en même temps.

Q.I.4) - Montrez qu'un processus peut entrer en section critique lorsque c'est possible, c-à-d lorsqu'elle est libre.

Q.I.5) - Que vient-on de montrer avec ces 2 propriétés ?

Q.I.6) - Donner un inconvénient à l'algorithme.

II Gestion d'ordre avec des sémaphores

On considère un système où s'exécutent trois processus (légers ou lourd) P1, P2 et P3 qui ont les caractéristiques suivantes :

- P1 exécute en boucle les tâches A puis B ;
- P2 exécute en boucle les tâches U puis V ;
- P3 exécute en boucle la tâche X.

De plus, les contraintes suivantes doivent être respectées :

- La tâche A de P1 produit un élément nécessaire à la tâche X de P3. Cela signifie qu'une occurrence de X ne peut pas démarrer avant la fin d'une occurrence de A.
- Les tâches B et U sont en exclusion mutuelle.

On note dA_i et fA_i respectivement le début et la fin de la tâche A. On fait de même pour toutes les tâches. Répondez aux questions suivantes :

Q.II.1) - Les ordres d'exécutions suivant sont-ils possibles sinon, quelles parties posent problème :

1(a) - $dA_1 fA_1 dX_1 dB_1 dU_1 fX_1 fU_1 fB_1 dA_2 dX_2 dV_1 fV_1 fX_2 fA_2 dU_2 dB_2 fU_2 fB_2 dA_3 fA_3 dB_3 fB_3$

1(b) - $dA_1 fA_1 dX_1 dB_1 fB_1 dA_2 dU_1 fA_2 fX_1 fU_1 dX_2 dV_1 fV_1 fX_2 dU_2 fU_2 dB_2 fB_2 dA_3 fA_3 dB_3 fB_3$

Q.II.2) - Donnez le graphe de précédence et d'exclusion mutuelle.

Q.II.3) - Gérez le problème entre P1 et P2 avec des sémaphores.

Q.II.4) - Gérez le problème entre P1 et P3 avec des sémaphores.

Q.II.5) - Peut-on utiliser la ou les mêmes sémaphores pour les questions II.3 et II.4 ?

III Producteur consomateur

Le problème du producteur consommateur est un problème classique de synchronisation en programmation multi-thread. Par exemple, le problème du producteur/consommateur présente un ensemble de threads « producteurs » qui dialogue avec un ensemble de threads « consomateurs » qui dialoguent grâce à une file de données partagées. On peut par exemple envisager un thread « Maître » qui reçoit les connexions des clients et qui envoie les sockets de discussions à des threads « Esclaves » qui traitent leurs demandes.

Pour éviter les problèmes d'accès concurrents à la liste de sockets il faut protéger cette donnée. Le but de l'exercice est de programmer la liste sous la forme d'un moniteur.

Pour les questions suivantes, dans un premier temps, vous ne donnerez que les algorithmes.

Q.III.1) - Quelles fonctions doit implémenter le moniteur ? Quelles sont celles qui doivent être protégées ?

Q.III.2) - Donnez la description de la structure de données qui permet de stocker cette file.

Q.III.3) - Donnez l'algorithme des fonctions qui permettent d'assurer l'ajout et le retrait d'une tâche (la tâche sera un simple int).

Q.III.4) - Modifiez ces fonctions de manière à assurer l'attente en cas de file pleine ou vide.

Q.III.5) - Donner l'implémentation de ces fonctions avec la librairie pthread. N'oubliez pas les fonctions d'initialisation et de libération de ressources.