Série d'exercice #7

Solution

IFT 2245

20 février 2019

7.1 Famine

Comment un système peut-il détecter que certain de ses threads sont en état de famine, ou sinon peut-il éviter ou tenter de gérer ce problème ?

7.1 Famine (Solution)

Comment un système peut-il détecter que certain de ses threads sont en état de famine, ou sinon peut-il éviter ou tenter de gérer ce problème ?

Solution

Il suffit alors de définir un maximum de temps T acceptable de non exécution pour un proccesus. Lorsqu'un processus demande une ressource, on initialise alors un *timer* qui lancera une interuption après un temps T si le processus n'a toujours pas obtenue la ressource demandée. On considère alors que le processus est en famine.

Un système peut tenter d'allouer les ressources sur la base du temps attendu par un processus. Les processus qui ont attendu plus longtemps devraient être priorisés.

Un système moins strict peut simplement garantir qu'un processus en famine se verra allouer des ressources en priorité par rapport à un processus qui n'est pas en famine.

7.2 État sûr et interblocage

Développer un scénario où un système commence dans un état *unsafe* et où malgré cela, l'exécution des threads termine sans rencontrer d'interblocage.

7.2 État sûr et interblocage (Solution)

Développer un scénario où un système commence dans un état *unsafe* et où malgré cela, l'exécution des threads termine sans rencontrer d'interblocage.

Solution

Soit l'état où il n'y a pas assez de ressource pour que tout les threads terminent. Un des threads termine sans avoir besoin de toutes les ressources demandés et libère ces ressources de telle sorte que toutes les autres threads peuvent finir.

7.3 Lockfree 1

Cliquer ici pour le code

1. Trouver les conditions de course présentes si le code est utilisé dans une application à plusieurs *threads*.
2. Corriger ces conditions de course en ajoutant les verrous et opérations correspondantes nécessaires. Décrire clairement quelles données sont protégées par chaque verrou.
3. Corriger ces mêmes conditions de course sans utiliser de verrous, en utilisant à la place une approche de synchronisation optimiste, en utilisant l'opération `compare&swap`.

7.3 Lockfree 1 (Solution)

Cliquer ici pour le code

1. Trouver les conditions de course présentes si le code est utilisé dans une application à plusieurs *threads*.
2. Corriger ces conditions de course en ajoutant les verrous et opérations correspondantes nécessaires. Décrire clairement quelles données sont protégées par chaque verrou.
3. Corriger ces mêmes conditions de course sans utiliser de verrous, en utilisant à la place une approche de synchronisation optimiste, en utilisant l'opération `compare&swap`.