DS 2012 Systèmes avancés

Damien Crémilleux

14 mai 2014

1 Questions courtes

1.1 Mémoire virtuellement partagée

Dans le cas d'une gestion de mémoire avec invalidation ; que se passe-t-il lorsque deux processus situés sur des processeurs différents accèdent régulièrement à une même variable ? Que peut-on faire pour améliorer l'efficacité dans ce cas ? Lorsque deux processus situés sur des processeurs différents accèdent à une même page, cela va provoquer un défaut de page pour le processus arrivant en deuxième. Si cela arrive régulièrement, les performances peuvent donc être dégradée. Afin de ne pas avoir ce problème et de garder une mémoire cohérente, il est possible de mettre en place le *multiple-writer protocol*. Ce protocole permet à plusieurs processus d'écrire en même temps sur une même page, si les modifications ont des localisations différentes.

1.2 Systèmes de fichiers distribués

Quel est l'intérêt d'avoir un serveur de fichier distribué stateless ? L'intérêt d'avoir un serveur de fichier stateless (comme NFS v3) est la résistance aux problèmes de réseau. Ainsi le serveur peut être situé dans un environnment instable, subir un crash et redémarrer sans problème.

> When NFS was developed there were many problems with the stability of networks and servers. NFS was designed to be an efficient protocol that could successfully handle an unstable environment without causing harm to the client machine. To operate in such a potentially harsh environment, NFS was built to be stateless. Being stateless means that there is no "state" that any operation could hold NFS in. For instance, in NFS version 2, when you perform a write, it will not return a result until the data has been written onto permanent storage. If the write never returns, it is up to the client to decide to continue waiting for the write to finish, or to not make the write.
>
> So being stateless means that the server can crash and the client will wait for the server to come back up. However, for some technical reasons as we'll explain below, there can be minor problems. The most notable one is the "Stale File Handle" error that can be seen after a server is rebooted. The cause of this one is, related to the file handles that the server uses with its filesystem. If those should change, then NFS mounts that the client will not have the correct filehandle and it will need to be unmounted and then remounted.

Quelles sont les limitations de NFS pour une utilisation dans un cadre massivement parallèle ? NFS a été désigné pour des petites infrastructures et présente donc des limites à cause de l'approche client/serveur. Ainsi, face à un grand nombre de client, le serveur NFS se trouvera surchargé.

1.3 Virtualisation

Quelles sont les différences entre les virtualisations de type I et II vues en cours, leurs avantages et inconvénients ? Il existe deux types de virtualisations :

- Le type I. L'hyperviseur et la machine virtuelle tourne directement au dessus du système d'exploitation. Le type I offre de meilleur performance car il y a un accès direct au matériel. Les inconvénients sont la modification du noyau, la complexité du système et le fait que ce ne soit pas une virtualisation complète du système.
- Le type II. L'hyperviseur et la machine virtuelle tourne au sein du système d'exploitation hôte. Ce type de virtualisation est plus axé sur le développement, car les machines virtuelles sont facilement récupérable après un crash. En outre, ce système est plus simple et ne recquiert pas de modification du noyau.

Différence entre para-virtualisation et virtualisation complète En para-virtualisation, le système d'exploitation tournant est modifié pour augmenter les performances. Le système est donc conscient d'être virtualisé. AU contraire, en virtualisation complète, le système d'exploitation n'est pas modifié, et n'a donc pas conscience d'être virtualisé.

1.4 MPI

Pourquoi ne peut-on pas partager de variables en MPI ? On ne peut pas partager de variables en MPI car il s'agit seulement d'une bibliothèque de passage de message, pour des machines distribuées. Il n'y a donc pas de mémoire partagée et de variable partagée.

Pourquoi n'utilise t'on pas le rendez-vous pour la communication en MPI ? En MPI, les communications sont non-bloquantes afin d'être plus efficaces. En outre, cela évite le risque de *deadlock* si un message se perd sur le réseau.

2 Exercice 1 : Interblocage

Ce document représente la politique de priorité du robot *Mars PathFinder*, qui a atterri sur Mars le 4 juillet 1997. Seulement, quelques jours après le début de sa mission, le robot fit de façon répété des *reset*, empêchant le bon déroulement de sa mission.

On peut constater sur le document que *Mars Pathfinder* contient un bus d'information (*Task bcdist*) qui permet de transmettre des informations entre différents composants du robot. La priorité associé à cette tâche est la plus

élevé. En outre, l'accès à ce bus est synchronisé à l'aide d'une exclusion mutuelle (mutex).

La récupération des données météorologiques (*Task ASI/MET*) est effectuée avec un thread de priorité basse. Quand les données doivent être envoyées, le thread utilisa le bus d'information. Pour cela, il acquiert un mutex, envoie les informationes sur le bus et relâche le mutex. Si jamais le thread se fait interrompre avant la fin de la communication, cela bloque le mutex jusqu'à ce que le thread reprenne et libère le mutex. On voit aussi que *Mars PathFinder* contient d'autres thread, de priorité intermédiaire.

Il est donc possible qu'une interruption arrive lorsque la tâche...

```
Pathfinder contained an "information bus", which you can think of
    as a shared memory area used for passing information between
    different components of the spacecraft. A bus management task
    ran frequently with high priority to move certain kinds of data
     in and out of the information bus. Access to the bus was
    synchronized with mutual exclusion locks (mutexes).

The meteorological data gathering task ran as an infrequent, low
    priority thread, and used the information bus to publish its
    data. When publishing its data, it would acquire a mutex, do
    writes to the bus, and release the mutex. If an interrupt
    caused the information bus thread to be scheduled while this
    mutex was held, and if the information bus thread then
    attempted to acquire this same mutex in order to retrieve
    published data, this would cause it to block on the mutex,
    waiting until the meteorological thread released the mutex
    before it could continue. The spacecraft also contained a
    communications task that ran with medium priority.

Most of the time this combination worked fine. However, very
    infrequently it was possible for an interrupt to occur that
    caused the (medium priority) communications task to be
    scheduled during the short interval while the (high priority)
    information bus thread was blocked waiting for the (low
    priority) meteorological data thread. In this case, the long-
    running communications task, having higher priority than the
    meteorological task, would prevent it from running,
    consequently preventing the blocked information bus task from
    running. After some time had passed, a watchdog timer would go
    off, notice that the data bus task had not been executed for
    some time, conclude that something had gone drastically wrong,
    and initiate a total system reset.
```

3 Exercice 2 : MPI