Série d'exercices #12

IFT-1215

April 3, 2015

18.6

L'algorithme d'ordonnancement multi-niveaux à rétroaction (*multi-level feedback queue scheduling*) ressemble à du FIFO au niveau le plus élevé et à du *round-robin* au niveau le plus bas, pourtant il se comporte souvent mieux que chacun des deux en terme des objectifs de performance. Expliquer pourquoi.

18.29

Qu'est-ce qu'un système *temps-réel*? Discuter l'impact d'un système temps-réel sur la conception du système d'exploitation, tels que les algorithmes utilisés et les fonctionnalités offertes.

18.39

Soit un système à C processeurs qui doit gérer l'exécution de P processus, où $P > C$. Considérer deux choix de l'ordonnanceur: soit il peut permettre à n'importe quel processus de s'exécuter sur le premier processeur libre, soit il peut limiter chaque processus à toujours s'exécuter sur le même processeur. Discuter des avantages et inconvénients de chacune des deux options.

18.17

Expliquer pourquoi ajouter de la mémoire DRAM peut augmenter substantiellement la performance d'un système.

18.20

Un manuel pour un système d'exploitation qui fut populaire mentionne que le nombre d'utilisateurs que peut gérer une machine augmente si les utilisateurs utilisent les mêmes programmes. Quelle caractéristiques de la mémoire virtuelle explique ce phénomène?

Segmentation et fragmentation

Une "alternative" à la *pagination* est la *segmentation*: par exemple chaque adresse logique de 32bit est décomposée en 8 bit de *numéro de segment*, et 24bit d'*offset* dans ce segment. Le processeur avait ensuite une table qui indiquait pour chaque segment son adresse physique de base et sa taille.

Donc pour l'adresse logique `SSoooooo` (où les S et les O sont des chiffres hexadécimaux correspondant au numéro de segment et à l'offset), le processeur calcule l'adresse physique de la manière suivante:

$$AP = \begin{cases} \textsc{Error} & \text{si } oooooo \geq SegmentSize[SS] \\ SegmentBase[SS] + oooooo & \text{sinon} \end{cases}$$

Chaque création de processus crée un segment pour le code, un autre pour la pile, et le processus peut par la suite allouer ou détruire des segments au besoin.

Jouez le rôle du système d'exploitation qui doit faire fonctionner les processus suivants sur une machine avec 16MB de mémoire et donc décider où placer ces segments dans la mémoire physique, *sans savoir à l'avance quelles requêtes vont être faites*. Estimer le coût de chaque genre d'opération, et essayer de minimizer le coût total d'exécution.

La séquence d'événements est la suivante:

- Création de P1 avec 1MB de code et $\frac{1}{2}$MB de pile.
- Création de P2 avec 1MB de code et $\frac{1}{2}$MB de pile.
- P1 alloue un segment de 6MB.
- P1 alloue un segment de 128KB.
- P1 libère le segment de 6MB.
- P1 alloue un segment de 12MB.
- P2 alloue un segment de 128KB.
- P1 libère son segment de 12MB.
- P2 alloue un segment de 256KB.
- P1 alloue un nouveau segment de 12MB.
- P2 alloue un segment de 64KB.
- P1 libère son segment de 12MB.
- P1 alloue un segment de 7MB.
- P2 alloue un autre segment de 64KB.
- P1 termine.
- P2 alloue un segment de 128KB.
- P2 libère son segment de pile.
- P1 est relancé et ré-exécute la même séquence d'allocations/désallocations.