Université de Strasbourg
Département d'Informatique

Année 2008/2009
L3 Informatique

Fondements des Systèmes d'Exploitation
Examen de première session - 8 janvier 2009

Durée : 2h.
Documents, calculatrices, portables interdits.
Le barème donné est indicatif.

Exercice 1 - Ordonnancement et threads (8 points)

Vous répondrez à ces questions en utilisant le langage C, sans oublier de le commenter. La seule fonction système à votre disposition est `malloc()`.

On considère un système d'exploitation gérant des processus et des threads. L'ordonnanceur, préemptif, ordonnance les threads uniquement, à l'aide d'un système de file d'attentes par priorité à 32 niveaux de priorité. Pour simplifier l'implémentation, vous pouvez considérer qu'un processus est simplement un thread "leader" d'un groupe de threads.

Lorsqu'un thread rend la main à l'ordonnanceur suite à l'expiration de son quantum ou un appel système, sa nouvelle priorité est calculée selon la formule suivante. `DeltaPri` est une fonction externe qui va affecter plus ou moins de priorité au thread concerné, selon différents paramètres - ne vous en souciez pas ; chaque thread dispose d'un niveau de priorité (*nice-level*).

```
pri = nicelevel + DeltaPri(threadID)
```

De plus, les threads devenus inactifs suite à un appel système, sont placés dans une file d'attente séparée *en attente de ressource*, jusqu'à ce que l'appel système termine.

1. Proposez une structure de données contenant les informations relatives aux processus/threads et à leur ordonnancement. Donnez également la structure de données des 32 files d'attentes par priorité, et de la file d'attente des threads en attente de ressources.

2. Écrivez la fonction appelée lorsqu'un thread rend la main à l'ordonnanceur, qui met à jour la priorité d'un thread et qui le déplace dans la file d'attente de threads en attente de ressources.

3. Écrivez la fonction qui déplace le thread de cette file d'attente vers la file d'attente des threads ordonnançables.

4. Lorsqu'un nouveau processus est créé, il n'a qu'un thread, qui hérite du nice-level du thread qui l'a créé. Écrivez la fonction qui crée ce processus, le thread associé, et le place dans la file d'attente d'ordonnancement adéquate.

Exercice 2 - Pagination (5 points)

L'algorithme de choix d'une page mémoire à remplacer et un élément très important pour l'efficacité du système de remplacement de pages dans le contexte du swapping. Cependant, il n'existe pas d'algorithme optimal implémentable dans un système d'exploitation généraliste où l'on ne connait pas à l'avance les pages qui seront accédées.

Voici un exemple d'accès à des numéros de page mémoire :

```
3  2  1  0  3  2  4  3  2  1  0  4
```

1. Montrez, à l'aide de cet exemple, que l'algorithme FIFO avec 3 cadres mémoires peut donner de meilleurs résultats que ce même algorithme avec 4 cadres mémoire (c'est l'*anomalie de Belady*).

2. Montrez que l'algorithme de remplacement LRU (*least recently used*) peut être moins efficace que FIFO à l'aide de ce même exemple.

3. Quel est le nombre de défauts de pages minimal qu'il pourrait y avoir avec cet exemple, lorsque la mémoire est composée de 3 et de 4 cadres ?

Exercice 3 - Questions diverses (7 points)

Répondez à ces questions en quelques lignes (*maximum* 8 lignes par question).

1. Quel est le rôle d'un système d'exploitation généraliste comme linux ? Comment parvient-il à assumer ce rôle ?

2. Quels sont les composants d'un système d'exploitation ? Précisez en particulier ce que sont le *bottom-half* et le *top-half*.

3. Qu'est-ce qu'un changement de contexte ?

4. Quelle est la différence entre un driver (*pilote de périphérique*) en mode caractère et un driver en mode bloc ?

5. Qu'est-ce qu'un filesystem ?

6. Un processus en cours d'exécution voit son image mémoire découpée en trois segments : TEXT, DATA et STACK. Quelle est l'utilité de la segmentation ?

7. À quoi sert le flag *copy-on-write* placé parfois sur une page mémoire accédée par un processus ? Dans quel cas est-il utilisé ?