Système d'exploitation

Examen – 28 mai 2013
2h

Le barème est donné à titre indicatif. Les différentes parties sont totalement indépendantes.
L'examen est volontairement trop long, passez à une autre partie si vous êtes bloqués.

I. Questions d'échauffement *8 points*

Les sous-questions suivantes sont indépendantes.

a) Comment les appels-système contribuent-ils à améliorer l'équité en alternant l'exécution des différents processus ? Comment fait-on si aucun appel-système n'est effectué ? *2 pts*

b) Si une page en *copy-on-write* ne contient que des zéros, que se passe-t-il quand un processus écrit à nouveau un zéro dedans (et donc ne modifie pas son contenu) ? Pourquoi ? *1 pt*

c) Quel est l'intérêt de l'appel-système vfork ? Que pourrait-il se passer si le père reprenait son exécution avant que le fils n'a fait exec ou exit ? *2 pts*

d) Comment implémenter un logiciel « preload » qui fait en sorte que les programmes les plus couramment utilisés soient chargés en mémoire (mais pas lancés) au démarrage de la machine (sans provoquer de swap génant) ? *3 pts*

II. Read/Write Locks *5 points*

On considère un verrou qui peut être pris de deux façon différentes :

- avec la fonction read_lock() (et read_unlock() pour le relâcher). Plusieurs personnes pourront tenir le verrou ainsi **simultanément**.
- avec write_lock() (et write_unlock()). Si quelqu'un tient le verrou avec write_lock(), personne d'autre ne peut le prendre, ni avec write_lock(), ni avec read_lock().

a) Quelle est la différence entre ce nouveau verrou et un verrou classique ? Quel est l'intérêt de ce genre de verrou ? Dans quel cas va-t-on recommander le nouveau verrou au lieu des verrous classiques ? *2 pts*

b) Avec une implémentation simple de ce nouveau modèle, que peut-il se passer si, toutes les secondes, une nouvelle personne prend le verrou avec read_lock() et ne le relâche avec read_unlock() que 2 secondes plus tard ? Comment y remédier ? *3 pts*

III. Table de pages *8 points*

On dispose d'une architecture 64bits avec 3 niveaux de table de pages matériels et des pages de 64ko (2^{16} octets). La table est constituée de tableaux de la taille d'une page. Ces tableaux peuvent être remplis avec des pointeurs vers les sous-niveaux (pointeurs de taille 8 octets) ou avec des PTE (de taille 32 octets).

Pour les questions suivantes, on pourra arrondir les calculs.

Rappel: $1\text{ k} \cong 10^3 \cong 2^{10}$ $1\text{ M} \cong 10^6 \cong 2^{20}$ $1\text{ G} \cong 10^9 \cong 2^{30}$ $1\text{ T} \cong 10^{12} \cong 2^{40}$ $1\text{ P} \cong 10^{15} \cong 2^{50}$

a) Rappelez rapidement ce qui est stocké dans chaque niveau de la table de pages. Pourquoi utilise-t-on en général des tableaux de la taille d'une page ? *1 pt*

b) Combien de pointeurs ou PTE peut-on mettre par page ? En déduire le nombre de bits utilisés pour chaque niveau de la table de pages. Combien de bits sont nécessaires pour décrire un décalage dans une page ? *2 pts*

c) En déduire la taille maximale des adresses virtuelles manipulées sur cette architecture. Comment relier ce résultat au fait que l'architecture est présumée « 64bits » ? Que fait-on des bits restants ? *2 pts*

d) Quel espace mémoire est nécessaire pour stocker la table des pages d'un processus qui utilise l'intégralité de son espace d'adressage ? Et s'il n'utilise que 100 octets ? Et s'il utilise 64Mo contigus ? Comparez avec le cas d'une table linéaire (non-hiérarchique à 1 seul niveau). *3 pts*

IV. Pagination à la demande *9 points*

a) Expliquer rapidement à quoi sert le pseudo code suivant ? Quand est-il appelé ? Dans quel contexte d'exécution ? A quelles lignes correspondent la traditionnelle erreur de segmentation (*Segmentation Fault*) ? *4 pts*

```
1     traitant(adresse, processus, en_ecriture)
2           trouver la zone mémoire du processus contenant adresse
3           si zone invalide
4                 tuer le processus
5           si zone valide
6                 si page absente
7                       allouer une page
8                       si page remplie préalablement
9                             lire la page depuis le disque
10                      mettre page dans table de pages du processus
11                otherwise si page présente
12                      si page en lecture seule et en_ecriture
13                            tuer le processus
```

b) Pourquoi la ligne 7 est-elle plus compliquée qu'il n'y paraît ? Quelle opération coûteuse peut-on avoir à faire s'il n'y a plus de mémoire disponible ? Combien cela couterait-il ? Et si cela échoue ? *3 pts*

c) Quels cas peuvent se présenter pour la page *remplie préalablement* ? Dans quel cas aura-t-elle été remplie par quelqu'un d'autre ? *2 pts*

V. Déduplication dans les systèmes de fichiers *10 points*

On considère un système de fichiers qui souhaite que des blocs identiques ne soient pas stockés plusieurs fois sur le disque. On parle de *déduplication* quand un contenu identique est stocké une seule fois physiquement (par exemple partagé entre plusieurs fichiers) au lieu d'être dupliqué en plusieurs endroits du stockage physique.

a) On s'intéresse tout d'abord au cas de fichiers *totalement* identiques. Si le système de fichiers ne supporte pas la déduplication automatique, que peuvent faire l'utilisateur ou l'administrateur pour ne pas dupliquer le contenu de plusieurs fichiers totalement identiques ? Quelles sont les limites de ces solutions ? *2 pts*

On s'intéresse maintenant aux sous-parties de fichiers, et pas seulement à des fichiers totalement identiques. Par exemple, si un grand fichier contient plusieurs fois le contenu d'un autre petit fichier, le contenu de ce petit fichier ne serait stocké qu'une fois sur le disque, et le grand fichier pointerait dessus.

b) Comment le stockage habituel des fichiers sous forme de blocs sur un disque va-t-il nous gêner ? *1 pt*

La déduplication se base généralement sur des fonctions de hashage du contenu des blocs (somme de contrôle, CRC, ...). On considère ici des blocs de 512 octets et des hashs de 32 bits.

c) En supposant avoir en mémoire tous les hash de tous les blocs disque, quels appels-système doivent être modifiés pour éviter la duplication des blocs identiques et comment ? *2 pts*

d) Si le processeur sait hasher un bloc à la moitié du débit mémoire habituel, combien de temps cela prend-il environ ? Comparez-le avec le coût d'un accès disque habituel. *1 pt*

e) Quelle est la probabilité que deux blocs soient identiques ? Quelle est la probabilité que deux blocs aient le même hash ? Expliquer ce qu'on doit faire pendant une écriture dans les 3 cas possibles, estimez les temps nécessaires par rapport à une écriture normale, puis déduisez-en l'impact moyen de la déduplication sur les performances d'une écriture. *4 pt*