Système d'exploitation

Examen – 19 mai 2009
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 *5 points*

Les sous-questions suivantes sont indépendantes.

a) Quels sont les objectifs principaux des systèmes d'exploitation ? On distinguera les intérêts des utilisateurs, des développeurs et ceux des administrateurs. *3 pts*

b) Dans un système d'exploitation préemptif, comment garantit-on qu'un processus en cours d'exécution va de temps en temps rendre la main au système et aux autres processus ? En quoi les appels-système interviennent-ils ? *2 pts*

II. Appels-système *8 points*

a) Expliquez rapidement le fonctionnement d'un appel-système. Qui décide de quelles fonctions appeler ? Quand et par qui sont-elles appelées ? Quels événements matériels sont impliqués ? *3 pts*

b) Que doit faire l'application passer les arguments de l'appel-système au système ? Comment récupère-t-on la valeur de retour ? Qui décide de tout cela ? *2 pts*

c) Expliquez rapidement comment ajouter un appel-système qui prend en paramètre une suite de 2 entiers en entrée et renvoie le suivant en retour. Par exemple passer 3 et 5 en entrée fait renvoyer 7. On négligera bien évidemment les aspects mathématiques pour se concentrer sur l'implémentation. Expliquer ensuite comment étendre l'appel-système précédent pour prendre en entrée une suite de 123 entiers et renvoyer les 7 suivants en retour. *3 pts*

III. Concurrence et synchronisation *5 points*

On considère le système de verrouillage suivant :

- Un verrou peut être pris par une des deux fonctions *ReadLock* et *WriteLock*. Il doit être relaché plus tard par le *ReadUnlock* ou *WriteUnlock* correspondant.
- *ReadLock* ne peut prendre le verrou immédiatement que si aucun *WriteLock* ne l'a pris sans l'avoir relâché. Sinon, *ReadLock* doit attendre.
- *WriteLock* ne peut acquérir le verrou que si aucun *ReadLock* ni aucun *WriteLock* ne l'a pris sans l'avoir relâché.

a) De quel genre de verrou s'agit-il ? Quel différence de comportement doit-il y avoir entre les tâches utilisant *ReadLock+ReadUnlock* et celles utilisant *WriteLock+WriteUnlock* ? *2 pts*

b) Quel problème peut se produire si de nombreux tâches prennent régulièrement le verrou avec *ReadLock* ? Comment appelle-t-on ce phénomène ? Que dire de l'équité dans ce modèle de verrouillage ? *1 pts*

c) Comment modifier l'implémentation pour y remédier ? Quel nouveau problème peut alors se produire ? Que dire maintenant de l'équité ? *2 pts*

IV. Gestion mémoire *10 points*

On considère une architecture où la taille des pages est 4ko.

a) On lance un programme qui alloue 4Mo (mapping privé anonyme) puis les remplit de zéros. On remarque que le lancement de notre programme provoque environ mille défauts de page. Expliquez à quoi sont dûs ces défauts de page. Expliquez ensuite pourquoi ces défauts de page disparaissent si le programme alloue la mémoire mais ne l'utilise pas. *2 pts*

b) On remarque ensuite que le lancement du processus, même sans allouer de la mémoire ou la modifier, provoque tout de même quelques défauts de page. A quoi sont-ils dûs ? Que peut-on observer si on lance plusieurs fois le processus d'affilée peu après le démarrage de la machine ? *2 pts*

c) On modifie maintenant notre programme pour se dupliquer par *fork* après avoir alloué les 4Mo de mémoire et les avoir remplis de zéros. Le fils créé par *fork* remplit alors les 4Mo avec d'autres valeurs. Qu'observe-t-on en terme de défauts de page ? Expliquez pourquoi et l'intérêt du modèle. *3 pts*

d) On modifie enfin le programme pour que le processus père remette les 4Mo de mémoire à zéro après que le fils a terminé. Qu'observe-t-on en terme de défauts de page ? Expliquez pourquoi. *1 pts*

e) Expliquez rapidement ce qui change dans les résultats précédents si on remplace les 4Mo de mapping privé par un mapping public (partagé). *2 pts*

V. Table de pages *7 points*

On suppose disposer d'une architecture 64bits avec 2 niveaux de table de pages matériel et des pages de 8ko (2^{13} octets). La table de pages est remplie avec des tableaux de pointeurs vers les sous-pages ou des PTE de taille 8 octets.

Pour les deux questions suivantes, on pourra arrondir les calculs.

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

a) Combien de bits sont nécessaires pour décrire un décalage dans une page ? 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. En déduire la taille maximale des adresses virtuelles manipulées sur cette architecture. Comment relier ce résultat au fait que l'architecture est dite « 64bits » ? *4 pts*

b) Si le système remplit l'intégralité de la table de pages d'un processus, quelle quantité de mémoire physique peut être nécessaire pour la stocker ? Quelles techniques peut-on utiliser pour réduire la mémoire physique nécessaire pour stocker une table de pages ? Et si la table des pages doit être entièrement remplie car le processus utilise intégralement son espace d'adressage ? *3 pts*

VI. Systèmes de fichiers *5 points*

a) Si on crée un nouveau fichier vide dans un système de fichiers Ext3, l'utilitaire **du** (*disk usage*) indique que ce fichier occupe 0 octets sur le disque. Que sont les *metadonnées* associées au fichier, et où sont-elles stockées ? *1 pts*

b) On écrit maintenant un caractère au début du fichier. Pourquoi les utilitaires **du** et **ls** indiquent-ils que l'*occupation disque* de ce fichier est différente de sa *taille* ? *1 pts*

c) On écrit maintenant un caractère un million d'octets plus loin dans ce fichier. Que devient la taille du fichier ? Que peut devenir son occupation disque ? Justifiez ce résultat en expliquant comment les données d'un fichier peuvent être stockées sur le disque. Qu'en est-il si on écrit un caractère mille milliards d'octets plus loin ? *3 pts*