Système d'exploitation

Examen – 22 mai 2012
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 *7 points*

Les sous-questions suivantes sont indépendantes.

a) Quelle est la différence entre l'administrateur (root) et le noyau ? Quels privilèges ont-ils ? Qui met en place ces privilèges ? 2 *pts*

b) Quel(s) avantage(s) et inconvénient(s) y a-t-il à avoir une *runqueue* (file de processus prêts) différente pour chaque processeur dans l'ordonnanceur du système d'exploitation ? 2 *pts*

c) Le noyau connaît les traductions d'adresses virtuelles en adresses physiques. Mais pourquoi doit-il tout de même manipuler des adresses virtuelles lorsqu'il veut accéder à une donnée en mémoire (déréférencer un pointeur) ? *1 pt*

a) Quel est la différence entre la taille et l'occupation disque d'un fichier ? Dans quel(s) cas sont-ils différents ? *2 pts*

II. Pagination à la demande *12 points*

On considère une architecture où la taille des pages est 4ko. La mémoire physique sera considérée comme infinie.

a) On lance un programme qui alloue 400ko (mapping public anonyme) puis les remplit de zéros. Expliquez pourquoi le lancement de notre programme provoque environ 100 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 400ko de mémoire et les avoir remplis de zéros. Le fils créé par *fork* remplit alors les 400ko avec d'autres valeurs. Quand le fils a terminé, le père remet la mémoire à zéro. Qu'observe-t-on en terme de défauts de page ? *2 pts*

d) Pour les questions suivantes, le mapping est maintenant privé et non plus public. Que devient votre réponse à la question précédente ? Expliquez ce qui se passe et l'intérêt du modèle. *3 pts*

e) On modifie ensuite le programme pour lancer deux fils se comportant comme ci-dessus. Expliquez les différents états des pages (protection matérielle, compteur de références, …) et les défauts de pages observés par les trois processus au fur et à mesure de leur exécution. *3 pts*

III. Table de pages *7 points*

On suppose disposer d'une architecture 64bits avec 3 niveaux de table de pages matériels et des pages de 16ko (2^{14} octets). La table de pages 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 16 octets).

Pour les 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) 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 ? *2 pts*

IV. Synchronisation passive *6 points*

On considère un système d'exploitation où les processus sont stockés dans une grande liste globale (liste simplement chaînée par un pointeur *next*). Chaque élément de liste contient le PID du processus et un pointeur vers le bloc de contrôle correspondant.

a) On s'intéresse à l'insertion d'un processus NEW d'identifiant PID au milieu de la liste (entre PREV et NEXT) implémentée de la façon suivante :

PREV->*next* = NEW ; NEW->*next* = NEXT ; NEW->*pid* = PID

Si une tâche regarde les différents PID dans la liste pendant l'insertion, quels sont les différents cas possibles et que dire de la cohérence de ce qu'elle va observer ? Comment modifier l'implémentation de l'insertion pour que tous les cas soient cohérents ? Que peut-on en déduire sur la nécessité de verrouiller l'insertion et/ou le parcourt de la liste ? *3 pts*

b) Mêmes questions pour la suppression de NEW entre PREV et NEXT :

NEW->*pid* = -1 ; NEW->*next* = NULL ; PREV->*next* = NEXT *2 pts*

c) Lors de la suppression de NEW, quand peut-on faire effectivement faire free(NEW) ? *1 pt*

V. Déduplication dans systèmes de fichiers *8 points*

On considère un système de fichiers qui souhaite que des blocs identiques ne soient pas stockés plusieurs fois sur le disque.

a) 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 entièrement identiques ? Quelles sont les limites de ce modèle, au niveau de l'application, du contenu et du système de fichiers ? *2 pts*

La déduplication se base généralement sur des fonctions de hashage du contenu des blocs (somme de contrôle, CRC, ...). Cette opération peut être supportée matériellement par les processeurs modernes. On considère ici des blocs de 512 octets et des hashs de 32 bits.

b) 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 et comment ? *2 pts*

c) Si le processeur sait hasher un bloc à la moitié du débit mémoire maximum, combien de temps cela prend-il environ ? En reprenant votre réponse à la question précédente, précisez la probabilité que chacun des cas possibles se produise et discutez des surcoûts et gains en temps. *2 pts*

d) Si le système stocke effectivement les hash en mémoire, que se passe-t-il après un reboot ? Si on envisage de stocker des hashs sur le disque, où serait-il judicieux de les stocker ? Réfléchissez à quand les hashs et les autres éléments du système de fichiers sont lus pour expliquer pourquoi votre solution est intéressante. *2 pts*