Système d'exploitation

Examen – 27 mai 2011
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) Il y a des parties du code du système d'exploitation qui sont portables. D'autres qui sont écrites spécifiquement pour une architecture matérielle. Donner des exemples dans le cadre de la gestion de la mémoire virtuelle et de l'ordonnancement des processus. Qu'est-ce que cela implique sur le langage de programmation utilisé ? *3 pts*

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

c) Dans un système d'exploitation préemptif, quel mécanisme permet de garantir que les processus actifs vont de temps en temps être préemptés de force par l'ordonnanceur ? Comment les appels-système interviennent-ils ? Quelle similarité y a-t-il avec le traitement des signaux ? 3 *pts*

II. Entrées-sorties ***7 points***

On considère un tube dans lequel un processus écrit pendant qu'un autre lit. Le système d'exploitation utilise un tampon mémoire intermédiaire comme intermédiaire.

a) S'agit-il d'entrées-sorties logiques ou physiques ? Pourquoi ? Comparez avec d'autres mécanismes de communication inter-processus. *2 pts*

b) Comment les données sont-elles transférées concrètement ? Quels accès mémoire dans quels espaces d'adressage sont nécessaires ? Dans quels contextes d'exécution se déroulent-ils ? 3 *pts*

c) Imaginez un mécanisme permettant de ne pas utiliser de tampon mémoire intermédiaire et donc de copier directement depuis la mémoire de l'écrivain dans celle du lecteur. *2 pts*

III. Sémaphores ***5 points***

On s'intéresse à une structure sémaphore et aux fonctions Post() et Wait() qui incrémentent/relâchent et décrémentent/prennent (attente passive) respectivement le compteur correspondant. Le système d'exploitation cible étant multiprocesseur avec ordonnanceur préemptif, on fera attention au verrouillage et aux interruptions.

a) En vous inspirant du TD2, expliquez comment implémenter cette structure et ces fonctions (en pseudo-code). *3 pts*

b) Discutez l'équité de votre implémentation selon la façon dont vous gérez file d'attente des processus endormis. Donnez un exemple concret de problème pouvant se poser. *2 pts*

IV. Table de pages *7 points*

On suppose disposer d'une architecture 64bits avec 4 niveaux de table de pages matériels et des pages de 2ko (2^{11} 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 deux questions suivantes, on pourra arrondir les calculs.

Rappel: 1 k = 10^3 = 2^{10} 1 M = 10^6 = 2^{20} 1 G = 10^9 = 2^{30} 1 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*

V. Pagination à la demande *13 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 1Mo (mapping privé anonyme) puis les remplit de zéros. Expliquez pourquoi le lancement de notre programme provoque environ 250 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 1Mo de mémoire et les avoir remplis de zéros. Le fils créé par *fork* remplit alors les 1Mo 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 ensuite le programme pour lancer trois 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 quatre processus au fur et à mesure de leur exécution. *3 pts*

e) On modifie ensuite le programme pour que le processus père remette les 1Mo de mémoire à zéro lorsque tous les fils ont terminé. Qu'observe-t-on en terme de défauts de page ? Expliquez pourquoi. *1 pt*

f) Expliquez rapidement ce qui change dans les résultats précédents si on remplace les 1Mo de mapping privé par un mapping public (partagé). *2 pts*