Série d'exercices 1 : Concepts généraux

1.1 Quelles sont les fonctions principales d'un système d'exploitation? Les interpréteurs de commandes et les compilateurs font-ils parties du système d'exploitation?

Le système d'exploitation gère et contrôle le matériel et offre aux utilisateurs une machine virtuelle plus simple d'emploi que la machine réelle (appels systèmes). Non, les interpréteurs et les compilateurs ne font pas parties du noyau du système d'exploitation.

1.2 Dans le système Linux, les appels système sont généralement effectués à partir de quelle couche: programme utilisateur, commande shell, procédure de la bibliothèque standard?

A partir de la bibliothèque standard des appels système, avec une instruction de déroutement (interruption logicielle, aussi connue sous le nom de « trap »).

1.3 Quelle est l'utilité de l'accès direct à la mémoire (en anglais Direct Memory Access DMA)? En supposant qu'une unité centrale de traitement serait capable de transferts à la même vitesse que l'unité DMA, serait-il néanmoins utile et souhaitable que le système d'exploitation utilise l'unité DMA?

L'unité DMA est spécialisé pour transférer (copier) des données entre un périphérique et la mémoire principale. Cela permet de libérer le processeur de telle sorte qu'il puisse exécuter une autre tâche. Cela permet au système de faire environ deux choses en même temps (la copie en arrière plan et l'exécution de tâche).

1.4 Les appels systèmes sont-ils exécutés en mode superviseur (noyau) ou en mode utilisateur?

Les appels systèmes sont demandés par les applications (en mode utilisateur) et sont exécutés dans le noyau (en mode système).

1.5 Il existe maintenant différents mécanismes pour interagir avec un ordinateur, qui présentent chacun une charge différente sur le système. Estimez la quantité de mémoire nécessaire pour mémoriser l'état d'un terminal alphanumérique (24 lignes de 80 caractères). Comment cela se compare-t-il à la mémoire requise pour l'état d'un terminal graphique (carte graphique, écran HD 1920x1080, clavier, souris)?

L'état minimal associé à un terminal consomme approximativement 24x80=1920 octets. Celui pour un affichage graphique couleur (24 bits par pixel) atteint 1920x1080x 3 octets = 6220800 octets.

1.6 Comment sont organisés les fichiers dans le système Linux? Un utilisateur peut-il accéder à un fichier d'un autre utilisateur? Si oui, comment?

Les fichiers sont organisés dans des répertoires. Chaque répertoire peut contenir des fichiers ou des répertoires (une structure arborescente). Pour contrôler les accès aux fichiers, chaque fichier a ses propres bits de permissions sur 9 bits (utilisateur, groupe et tous). Un utilisateur peut accéder à un fichier d'un autre utilisateur si les permissions lui permettent (membre du groupe ou les bits pour tous). Pour accéder au fichier, il faut utiliser le chemin d'accès est absolu.

1.7 Pour lancer en parallèle plusieurs traitements d'une même application, vous avez le choix entre les appels système fork() et pthread_create(). Laquelle des deux possibilités choisir? Pourquoi?

pthread_create() est à privilégierm car la création de process fork() consomme plus de mémoire (chaque processus a son propre espace) et prend aussi plus de temps à démarrer. Par contre, il faut faire attention au conflit d'accès aux objets partagés avec pthreads. Aussi, pour une raison de sécurité on pourrait privilégier des processus. Par exemple, en utilisant des processus pour chaque onglets dans un navigateur Web fait en sorte de rendre beaucoup plus difficile à un site Web malicieux d'accéder à la mémoire d'une autre page chargée (i.e. compte bancaire).

1.8 Citez quatre événements qui provoquent un changement de contexte (invocation de l'ordonnanceur) d'un processus en cours d'exécution dans le système Linux.

1) Fin d'un quantum, 2) Demande d'une E/S, 3) Terminaison du programme, 4) Mise en attente par un appel système bloquant, comme sem_wait() sur un sémaphore…

1.9 Lesquelles des opérations suivantes devraient-elles être permises seulement en mode système: a) désactiver les interruptions, b) lire l'horloge, c) changer l'heure de l'horloge, d) changer la table de pages?

Seule l'opération de lecture de l'heure (b) est permise en mode usager car elle ne met pas en péril l'intégrité du système ni ne révèle d'information potentiellement confidentielle.

1.10 Quel est le rôle de l'ordonnanceur? Comment peut-il favoriser les processus interactifs?

L'ordonnanceur gère l'allocation du processeur aux différents processus. Il alloue le processeur aux différents processus à tour de rôle pour un quantum de quelques millisecondes. On peut favoriser les processus interactifs en diminuant la priorité des processus qui font du travail en arrière-plan et en privilégiant les processus qui n'ont pas épuisé leur quantum avant de retomber en attente d'E/S.

1.11 Un système avec mémoire virtuelle contient de la mémoire cache, de 2ns de temps d'accès, de la mémoire vive, de 10ns de temps de pénalité d'accès (i.e.

10ns qui s'ajoutent aux 2ns d'essai d'accès en cache), et un disque, de 10ms de temps de pénalité d'accès. En moyenne, 95% des accès peuvent se faire en mémoire cache et, pour les accès qui ne peuvent être servis en cache, 99% d'entre eux peuvent se faire en mémoire vive. Si l'accès n'est pas en mémoire, alors le système d'exploitation doit faire un accès disque. Quel est le temps d'accès moyen à la mémoire dans cette situation?

Le temps d'accès moyen est donc de 2ns + .05 x 10ns + .05 x .01 x 10,000,000ns = 5002.5ns. Ceci montre bien l'importance d'avoir un taux de succès extrêmement élevé au niveau de la mémoire vive car chaque accès disque prend une éternité et affecte énormément la performance.

1.12 Dans quelles conditions le partage de données pose-t-il des problèmes dans un programme ayant plusieurs fils d'exécution?

Deux fils d'exécution qui modifient des données en mémoire en plusieurs opérations peuvent s'interférer et causer une condition critique. La valeur finale va dépendre de l'ordre exact des opérations. Il faut donc utiliser des instructions de synchronisation pour protéger les accès à ces données, par exemple avec un verrou.

1.13 Expliquez pourquoi les processeurs ont deux modes de fonctionnement (noyau et utilisateur).

A partir du moment où il faut isoler les utilisateurs les uns des autres, il faut un mode privilégié pour exécuter le système d'exploitation. Ceci empêche des programmes non-privilégiés de prendre le contrôle du système. Seul le système d'exploitation peut gérer le système et imposer les permissions.

1.14 Lors d'un appel système pour la lecture, le processus est normalement bloqué jusqu'à ce que la donnée soit disponible. Qu'en est-il pour les écritures, le processus est-il nécessairement bloqué jusqu'à ce que l'accès disque soit complété?

Sur la plupart des systèmes, les données à écrire sont mises en queue par le système d'exploitation et le contrôle revient à l'application sans bloquer. Ceci permet à l'application de poursuivre et est plus efficace. Si l'application veut être certaine que les données ont bel et bien été écrites sur le disque, et survivraient à une panne de courant impromptue, il existe des appels systèmes à cet effet (fsync).

1.15 Pour le programmeur, est-il utile de savoir si une fonction de la librairie standard résulte ou non en un appel système?

En termes de fonctionnalité, cela ne change habituellement rien. Toutefois, il est bon de le savoir pour une question de performance. Dans certains cas, il peut être important de savoir si cet appel peut résulter en un blocage. Aussi, chaque appel système implique un délai et si on fait des appels systèmes très

fréquemment, cela va ralentir le programme. Il faut donc s'assurer de faire des appels systèmes de manière judicieuse.

1.16 Quelle est la différence entre une interruption logicielle (aka trap, déroutement) et une interruption matérielle?

Une interruption matérielle est causée par un événement extérieur et est asynchrone. Par exemple, une touche sur un clavier déclenche une interruption matérielle. Une interruption logicielle découle de l'exécution du programme et arrive de manière synchrone. Par exemple, si le processeur rencontre une instruction invalide, le programme fait un accès à une adresse invalide, il se produit une division par zéro. Un appel système est aussi une sorte d'interruption logicielle. Le mécanisme utilisé pour traiter les interruptions est généralement le même qu'il soit matériel ou logiciel. Au démarrage de l'ordinateur, le système d'exploitation configure une table de gestionnaires d'interruptions avec des adresses de fonctions. Quand une interruption d'un certain type survient, le processeur exécute la fonction qui lui est associée.