Chapitre 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 (trap).

1.3 Quelle est l'utilité de l'accès direct à la mémoire (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 permet de libérer le processeur, de sorte qu'il puisse exécuter une autre tâche. Cela peut permettre au système de faire environ deux fois plus de travail dans le même temps.

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

Les appels systèmes sont effectués 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)? A la mémoire requise pour servir un client du site Web?

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. Le protocole HTTP est nominalement sans état, chaque client ne prend aucun espace après que la page Web lui ait été délivrée.

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 son propre code d'accès sur 9 bits. Un utilisateur peut accéder à un fichier d'un autre utilisateur si le code d'accès du fichier le permet. 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 ( ) car le fork( ) consomme beaucoup d'espace (duplication de processus). Mais il faut faire attention au conflit d'accès aux objets partagés.

1.8 Citez quatre événements qui provoquent l'interruption de l'exécution d'un processus en cours dans le système Linux.

1) Fin d'un quantum, 2) Demande d'une E/S, 3) Arrivée d'un signal, 4) Mise en attente par un appel système comme sem_wait sur un sémaphore…

1.9 Lesquelles 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 milisecondes. L'ordonnanceur peut favoriser les processus interactifs en rehaussant la priorité des processus qui font des E/S et en particulier qui n'épuisent pas 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 ne mémoire cache et, pour les accès qui ne peuvent être servis en cache, 99% d'entre eux peuvent se faire en mémoire vive. Quel est le temps d'accès moyen sur ce système?

Le temps d'accès moyen est donc de 2ns + .05 x 10ns + .05 x .01 x 10000000ns = 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 Pourquoi le partage de données pose-t-il des problèmes dans un système multiprogrammé en temps partagé?

Un autre processus peut accéder aux données partagées avant qu'un autre processus n'ait fini de les utiliser (modifier). Il faut donc utiliser des instructions de synchronisation pour protéger les accès à ces données, par exemple les sémaphores.

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 lui permet de gérer le système et de contrôler les permissions données à chaque utilisateur.

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 fonctionalité, cela ne change usuellement rien. Toutefois, cela peut donner une bonne idée de la performance à attendre. Dans certains cas, il peut être important de savoir si cet appel peut résulter en un blocage.

1.16 Quelle est la différence entre un déroutement (trap) et une interruption?

Une interruption est causée par un événement extérieur, asynchrone. Un déroutement découle de l'exécution du programme et arrive de manière synchrone (e.g. instruction invalide, accès à une adresse invalide, division par zéro, appel système...). Le mécanisme utilisé pour répondre aux deux est généralement le même, une table (interrupt vector) de gestionnaires (interrupt handler).