Examen IN201 - Cours de système d'exploitation

14 Janvier 2019

1. (1 point) Décrivez en quelques lignes les principaux rôles du système d'exploitation.

   Solution:

   - Faciliter l'usage de la machine, en fournissant des librairies et des couches d'abstractions facilitant l'écriture de programmes.
   - Permettre la coexistence de plusieurs tâches sur la machine, et gère les problèmes de concurrence, sécurité, et de partage de ressources que cela implique.

2. (1 point) Quels sont les principaux composants matériels d'un ordinateur?

   Solution:

   Le processeur, la mémoire, le bus, et les périphériques (carte réseau, disque dur, etc.)

3. (4 points) Décrivez la séquence des opérations faites par le processeur pour exécuter une instruction, lorsque la prochaine instruction à exécuter est de type `load r1, r2` (stocke contenu de la mémoire à l'adresse contenue dans `r2` dans le registre `r1`).

   Solution:

   1. Lecture mémoire de l'instruction à l'adresse du pointeur d'instruction;
   2. Décodage de l'instruction; accès au contenu de r_2.
   3. Demande (à travers le bus) de lecture mémoire à l'adresse r_2
   4. Récupération (par le bus) de la valeur contenue à l'adresse r_2 et écriture dans r_1.
   5. Mise à jour du pointeur d'instruction vers l'instruction suivante.

4. (3 points) Sous UNIX, l'appel système `read` permet de copier des données en provenance d'un file descriptor. Un file descriptor représente par exemple une connection réseau ou un fichier ouvert précédemment.

   (a) L'appel à `read` peut être bloquant, c'est à dire bloquer le thread qui l'appelle. Qu'est-ce que cela signifie?

   Solution:
   Cela signifie que le thread qui l'exécute peut passer dans l'état bloqué, i.e. ne peut plus être élu par l'ordonnanceur tant que la donnée qu'il attend n'est pas disponible.

   (b) À quelle famille de mécanisme de contrôle d'accès correspondent ces file descriptors?

Solution:
Il s'agit de capacité: le fait d'avoir un file descripteur dit au noyau qu'on a le droit d'aller lire dedans.

5. (4 points) On vous donne le programme suivant:

```
void main(void){ int b = 3; f(b); return g(b); }
void g(int c){ int i = 1; int j = i + c; if(j == 11) print_secret();
return; }
void f(int d);
```

   (a) En supposant qu'il n'y a pas d'optimisation, dessinez l'état de la pile avant l'instruction `if;` de `g`.

   (b) On suppose maintenant que le programmeur a oublié d'initialiser `i` (on supprime la partie soulignée). Écrivez une fonction `f` de manière à ce que `g` affiche le secret.

   Solution:
   La pile est `| b=3 | (program counter de retour) |c = 3 | i=1 | j=4 |`
   Pour f, il faut écrire une fonction qui laisse 8 à l'emplacement de `i` sur la pile, par exemple:
   `void f(int b){ int i = 8; int j; return; }`.

6. (4 points) Vous avez 6 gros programmes à exécuter sur 3 processeurs: A prend 70 minutes, B, C et D 60, E et F 40.

   (a) Quel est le temps minimal requis pour exécuter toutes les tâches?

   Solution:
   (70 + 60*3 + 40*2) /3 = 110 minutes

   (b) Quel algorithme permet d'exécuter toutes les tâches dans ce temps minimal?

   Solution:
   C'est l'algorithme de McNaughton

   (c) Dessinez un plan d'ordonnancement permettant d'exécuter toutes ces tâches dans le temps minimum.

   Solution:
   L'algorithme de McNaughton fournit la réponse.

   (d) Est-ce que ce plan d'ordonnancement fait des préemptions et/ou des migrations? Si oui, dites à quels moment et les tâches concernées.

   Solution:
   L'algorithme de McNaughton impose des migrations à plusieurs processus.

7. (3 points) Vous développez un site web permettant de catégoriser des musiques automatiquement à partir de fichiers MP3 qu'on vous envoie. Chaque requête d'un client est traitée en plusieurs étapes:

   1. Récupération du fichier du client à partir du réseau et mise en mémoire RAM.
   2. Calcul de la signature du fichier.

3. Récupération des informations sur la chanson correspondant à la signature dans une grosse base de donnée située sur le disque dur.
4. Envoi du nom de l'artiste au client sur le réseau.

(a) Quel est l'intérêt de séparer chacune de ces étapes en plusieurs threads? Autrement dit: quel problème aurait-on en traitant les requêtes dans un code séquentiel?

Solution:
La séparation en thread permet de mieux utiliser l'ordinateur (meilleur débit): pendant qu'un thread attend des entrées réseaux, un autre peut calculer la signature par exemple. Un code séquentiel bloquerait en attente du réseau dans la première étape, puis en attente du disque dur dans la troisième, etc.

(b) Expliquez par quel mécanismes du matériel et de l'OS, la séparation de ces différentes étapes en plusieurs processus, permet à une erreur de manipulation de pointeur dans l'étape 1 d'éviter d'affecter les autres processus.

Solution:
Le confinement mémoire, fourni par le système d'exploitation en paramétrant la MPU ou la MMU, permet d'éviter la propagation des erreurs d'accès mémoire.

(c) Est-ce que le processus correspondant à l'étape 1. doit avoir accès au système de fichier, et en particulier au fichier de base de donnée? Justifiez votre réponse.

Solution:
Non (suivant le principe du moindre privilège), car il n'en a pas besoin. Ne pas lui donner permet par exemple d'éviter que ce processus puisse corrompre la base de donnée en cas d'attaque.