Département Informatique

Promotion : 1998–2001	Année scolaire : 1999–2000
2ème année	Date : 3 mars 2000, 8h55–11h

Module SST 201

—

Systèmes d'exploitation et leur support d'exécution

—

Contrôle de connaissance de 2 heures [1]

Merci de répondre (au moins) dans les blancs.

1 Généralités système

Question 1 : À quoi sert un système d'exploitation (succinctement...) ?

Question 2 : Qu'est-ce que la multi-programmation ? En quoi cela permet une meilleure utilisa-

1. Sans document, sans calculatrice, sans triche, sans copie sur les voisins, sans micro-ordinateur portable ou non, sans macro-ordinateur, sans téléphone portable ou non, sans talkie-walkie, sans pompe, sans anti-sèche, sans tatouage ni vêtement imprimé en rapport avec le sujet, sans mouchoir de poche pré-imprimé,...

tion des ressources, de la ou des unités centrales ?

Question 3 : Quel est l'intérêt de la notion d'« utilisateur » ?

Question 4 : Quel est la différence entre un noyau et un micro-noyau ? Dans quelle catégorie se range Unix ?

Question 5 : Qu'est-ce que le contexte matériel d'un processus ?

Question 6 : Quels sont les mécanismes matériels nécessaires à la mise en œuvre du temps partagé ?

Question 7 : En quoi l'appel d'une fonction système par un processus utilisateur est différent de l'appel d'une fonction classique ?

2 Entrées-sorties

Question 8 : La routine de traitement des interruptions d'un ordinateur requiert 200 μs (temps de commutation des processus inclus) par impulsion d'horloge. La fréquence de l'horloge étant de 100 Hz, quel est le pourcentage de temps que le processeur consacre à l'horloge ?

Question 9 : Des requêtes arrivent, dans cet ordre, au pilote du disque dur pour les cylindres 10, 22, 20, 2, 40, 6 et 38. On néglige le temps d'accès aux secteurs dans ces cylindres et on ne s'intéresse qu'au temps d'accès aux cylindres. Le déplacement des têtes pour aller d'un cylindre à un cylindre voisin est de 2 ms. Les têtes sont initialement positionnées sur le cylindre 20. Quel est le temps de recherche pour les algorithmes suivants :

1° premier arrivé, premier servi ;

2° recherche la plus courte (la plus proche de la tête) d'abord ;

3° algorithme de l'ascenseur.

3 Gestion mémoire

Question 10 : Donner différentes façons permettant d'avoir des programmes ou processus dont la somme des tailles est supérieure à la taille de la mémoire.

Question 11 : Pourquoi offrir de la mémoire virtuelle sachant que Bill GATES a dit que 640 Ko suffisaient ?

Question 12 : Comment peut-on réaliser un système avec une pile utilisateur qui grossit à la

demande ?

Question 13 : Quel est le rôle d'un cache ?

Question 14 : Donner les endroits d'un ordinateur, d'un périphérique ou d'un système d'exploitation,... où des caches sont/peuvent être utilisés.

4 Systèmes de fichiers

Question 15 : Donner les avantages et les inconvénients respectifs des liens symboliques et des liens « durs » (*hard*) en Unix.

Question 16 : À quoi peut bien servir l'appel système Unix `mknod()` ?

Question 17 : Le protocole de gestion de fichiers à distance NFS (*Network File System*) est « sans état » côté serveur mais il existe aussi d'autres protocoles tels que le défunt RFS (*Remote File System*) qui conserve l'état de ses clients côté serveur. Quels avantages et inconvénients sont

associés à chaque vision du monde ?

5 Signaux

Question 18 : Quelle est la fonction *principale* de l'appel système `sigaction()` d'Unix SVR4 telle qu'elle a été vue dans le dernier TP sur Minix ? On rappelle l'usage :

```
#include <signal.h>
int sigaction(int sig, const struct sigaction *act, struct sigaction *oact)
```

et la structure `sigaction` contient les champs

```
void      (*sa_handler)();
void      (*sa_sigaction)(int, siginfo_t *, void *);
sigset_t  sa_mask;
int       sa_flags;
```

Question 19 : À quoi sert la commande Unix `kill` (outre le fait qu'elle permet de tuer un processus...) ?

6 Concurrence, parallélisme

Question 20 : Afin de programmer l'interaction entre le micro-contrôleur et le processeur de traitement du signal (DSP) d'un terminal GSM vous avez besoin de primitives atomiques de communications. Le choix de la solution matérielle a été fait par votre manager incompétent après réception d'un fort dessous de table. Le seul moyen de partage présent est une zone de mémoire commune et malheureusement la seule instruction atomique commune au micro-contrôleur et au DSP est l'instruction `INCR` *R*,*m* qui permet d'incrémenter[2] la case mémoire *m* et de mettre le résultat aussi dans le registre *R*.

2. C'est à dire « ajouter 1 à ».

Dans le but de ne pas perdre votre poste, vous devez sauver votre entreprise et proposer une solution pour permettre d'exécuter des sections critiques sur le micro-contrôleur sans être perturbé(e) par le DSP et réciproquement.

Valeur initiale de m :

Prélude de section critique :

Ici vous mettriez le code spécifique d'un section critique de votre application.

Postlude de section critique :

Vous avez réussi ? Si oui, votre manager aura une forte promotion mais pas vous. Si non, vous serez viré(e) pour incompétence et votre manager aura une moindre promotion. :-(

Question 21 : Pourquoi le concept de processus léger (*threads*) a-t-il été rajouté à celui de processus lourd ?

Question 22 : Quels sont les facteurs limitant l'efficacité des systèmes d'exploitation tournant

sur multi-processeurs à mémoire partagée avec l'augmentation du nombre de processeurs ?

Question 23 : Que fait l'appel système Unix `fork()` du point de vue utilisateur ?

Question 24 : Que fait l'appel système Unix `fork()` du point de vue noyau ?

Question 25 : Que fait un appel système Unix de type `exec()` du point de vue utilisateur ?

Question 26 : Que fait un appel système Unix de type `exec()` du point de vue noyau ?

Question 27 : Quand est-ce qu'un appel système Unix de type `exec()` retourne quelque chose ? Quelle est la difficulté que cela implique dans la réalisation au niveau système ?

Question 28 : Imaginer des scénarii possibles en cas d'un `fork()` au niveau utilisateur dans un processus multi-thread de niveau utilisateur (donc avec des *user threads*).

Question 29 : Imaginer des scénarii possibles en cas d'un `fork()` au niveau utilisateur dans un

processus multi-thread de niveau noyau (donc avec des *kernel threads*).

Question 30 : Dans le cas d'Unix, un processus est associé à 2 structures du noyau, `u` et `proc` :

- `u` contient les informations nécessaires au bon fonctionnement du processus lorsqu'il est actif (le noyau exécute un service pour ce processus) ;
- `proc` contient l'information sur le processus nécessaire en permanence (pour traiter une interruption matérielle, etc).

Donner sur la table 1 des exemples d'informations qui doivent se trouver dans l'une ou l'autre des structures en fonction des utilisations suivantes.

Question 31 : Expliquer en quoi consiste le mécanisme d'inversion de priorité.

TAB. 1 – *Contenu des tables* `u` *et* `proc`.

	Structure `u`	*Structure* `proc`
Gestion des fichiers		
Gestion de la mémoire, mémoire virtuelle		
Gestion des entrées-sorties, pilotes de périphériques		
Ordonnancement des processus		