UNIVERSITE DE NICE SOPHIA ANTIPOLIS

EXAMEN DE SYSTEME D'EXPLOITATION

MASTER 1 D'INFORMATIQUE

PR SID TOUATI

- **Durée : deux heures.**
- **Aucun document n'est autorisé.**
- **L'utilisation d'un périphérique informatique est interdite, à l'exception d'une calculette simple.**
- **Toute remise de copie après deux minutes de la fin du temps réglementaire sera sanctionnée par un malus.**
- **On veillera à être claire dans sa réponse, les calculs doivent être expliqués. Une réponse mal comprise est une réponse mal notée.**

EXERCICE 1 : GESTION DE LA MEMOIRE (7 PTS)

---

1. Quels sont les avantages de la mémoire virtuelle par rapport à un système de mémoire « réelle » ? 1

   Protection mémoire entre processus, partage transparent de la mémoire physique, avoir plus d'espace adressable.

2. Expliquez la différence entre la segmentation et la pagination, les intérêts de l'une par rapport à l'autre. 1

   La segmentation nécessite moins de circuiterie complexe dans le processeur, données contiguës en mémoire. Elle nécessite des registres. La mémoire accédée par le programme est contiguë (mieux pour les perfs).

   La pagination est mieux adaptée aux besoins actuels (multi processus, protection, swap). Elle nécessite une MMU.

3. A quoi sert le TLB dans un processeur ? (TLB= *Translation Lookaside Buffer*) 1

   Transformer rapidement l'adresse virtuelle en une adresse physique (TLB= table contenant les adresses physiques des pages mémoire).

4. Soit un ordinateur avec un processeur multi-cœurs. Le processeur contient 4 cœurs 64 bits. La taille de la mémoire physique est de 4 giga octets. La taille d'un cadre (page physique) est égale à celle d'une page mémoire, qui est de 8 ko. Il y a une seule mémoire centrale dans le système. La taille du fichier d'échange (*swap*) est de 6 giga octets. Décrivez et calculez ce qui suit : 1,5

   a. L'espace virtuel adressable par le processeur multi-cœurs. $2^{\text{taille\_adresse\_mémoire=}}\ 2^{64}$

b. Le nombre de pages physiques (cadres). Taille_mémoire/taille_cadre=4 Go/8ko=$4 \times 2^{30}/(8 \times 2^{10})=2^{19}$

c. Le nombre total de pages de la mémoire virtuelle pouvant être allouées sur le système. Taille_swap/taille_page=6 Go/8 ko= $¾ \times 2^{20}$

5. Supposons que le processeur 64 bits réserve 32 bits de l'adresse virtuelle pour définir le numéro de la page, et 32 bits d'adresse virtuelle pour définir le déplacement dans la page. Calculez : 1

a. La taille maximale d'une page $2^{\text{Taille\_adressage\_deplacement}}= 2^{32}$

b. Le nombre maximal de pages mémoire adressables par un processus $2^{\text{Taille\_adressage\_page}}= 2^{32}$

6. Expliquez ce que veut dire un accès mémoire aligné. Que se passe-t-il si un programme effectue un accès mémoire non aligné ? 1,5

Lorsque le programme veut accéder à une donnée de taille 2^a à une adresse qui n'est pas un multiple de 2^a. Selon les processeurs, une exception est levée (ou une erreur d'exécution).

EXERCICE 2 : PROCESSUS ET THREADS (7 PTS)

---

1. Soient les deux fonctions C suivantes : 2

```
int tab[10000] ;
void fonction1() {
 int i ;
  for(i=0;i<500;i++) {
    tab[i]=i;
  }
}
```

```
void fonction2() {
 int i ;
  for(i=500;i<10000;i++) {
    tab[i]=i;
  }
}
```

Les cases du tableau `tab` sont initialisées à zéro. Donnez le contenu du tableau `tab` après la fin d'exécution des deux fonctions dans les trois cas suivants :

a) Les deux fonctions s'exécutent dans deux processus différents (chaque fonction dans un seul processus chacune).

Il y a deux tableaux nommé tab non partagés entre les deux processus. Tab[i] contient donc la valeur de i, i allant de 0 à 499 dans le premier processus. Dans le 2e processus, Tab[i] contient donc la valeur de i, i allant de 500 à 9999

b) Les deux fonctions s'exécutent dans deux *threads* du même processus (chaque fonction dans un seul *thread* chacune).

Les deux threads partagent le même espace d'adressage ; Il y a donc un seul tableau nommé tab partagé. Tab[i] contient donc la valeur de i, i allant de 0 à 9999.

c) Les deux fonctions s'exécutent dans deux *threads* de deux processus différents (chaque fonction dans un seul thread de chaque processus).

Réponse identique au cas numéro (a)

2. Supposons que j'ai une machine avec deux processeurs, chacun ayant quatre cœurs

   a. Dans un premier cas, je suppose que j'ai six processus qui sont en état « prêt ». Quelle est d'après vous la politique d'ordonnancement de processus qui donnerait les meilleures performances pour le système ? Argumentez votre réponse. 1,5

      Etant donné qu'il y a plus de cœurs que de processus, n'importe quelle stratégie d'ordonnancement qui affecte entièrement un core à un processus donne de bons résultats. Il y a plus de ressources que de demandes.

   b. Maintenant, je suppose que j'ai douze processus triés du plus rapide au plus lent (le processus 1 est le plus rapide). J'opte pour un mécanisme d'exécution par lot (*batch*). 1,5

      (1) Que veut dire une exécution par lot ? Une exécution où le job est exécuté entièrement avant de passer au job suivant.

      (2) Le temps de complétion d'un processus est le temps entre la soumission du processus par l'utilisateur et la terminaison de l'exécution du processus. Quelle est la stratégie d'ordonnancement par lot qui engendre le temps moyen à complétion optimal ? Ordonnancement du plus court d'abord.

3. Dans les systèmes interactifs, une stratégie d'ordonnancement de processus connue est celle appelée ordonnancement par tourniquet (*round robin*); Elle utilise un *quantum* de temps pour exécuter un bout de chaque processus. Expliquez comment choisir une « bonne » valeur pour ce *quantum* (quel avantage d'avoir une petite valeur de *quantum* par rapport à une valeur plus élevée, et vice-versa). 1

   Un quantum avec une petite valeur offre plus d'interactivité mais le coût du changement de contexte devient important.

4. Dans mon système, j'ai deux utilisateurs appelés Pancho et Rancho qui doivent faire le même calcul informatique. Pancho crée dix processus différents et Rancho crée un seul processus mais qui lance dix threads différents. Expliquez dans quel type de système Pancho serait favorisé par rapport à Rancho, et inversement. 1

   Cette question n'a pas de réponse juste, elle permet au professeur d'évaluer l'esprit critique et de réflexion de l'étudiant. Le but de cette question est de pousser l'étudiant à émettre des hypothèses comme :

   - Pancho serait favorisé si l'ordonnanceur système partage le temps CPU de façon équitable entre les processus. Rancho serait favorisé, sur l'ordonnanceur système partage le temps CPU en donnant moins de priorité aux utilisateurs ayant beaucoup de processus.

- Si le système reconnaît les threads au niveau noyau, Rancho aurait peut être une exécution plus rapide car les threads sont ordonnancés comme des processus en parallèle mais avec un changement de contexte plus rapide.

- Si le système ne reconnaît pas les threads au niveau noyau, Pancho aurait peut être une exécution plus rapide car il profite du parallélisme des processus, alors que Rancho aura un seul thread qui s'exécute à la fois.

- etc

EXERCICE 3 : SYSTEMES DE FICHIER (6 PTS)

---

1. Le système de fichier `ext3` a introduit la journalisation dans Linux.

   a) Listez et expliquez les niveaux de journalisation d'`ext3`. 1

   Voir le TD numéro 3.

   Comment comparer entre les performances des différents niveaux de journalisation avec une démarche « scientifique » ? Voir le TD numéro 3.

   b) Quels sont les avantages et les inconvénients d'avoir des grandes ou des petites tailles de bloc disque. 1

   Petite taille de bloc : fragmentation limitée mais coût d'E/S plus élevé (temps de déplacement du bras disque).

   Grande taille de bloc : fragmentation plus élevée mais coût d'E/S plus diminué.

   c) Qu'est ce qui détermine le nombre maximal de fichiers et de répertoires dans une partition `ext3` ? 1

   Le nombre total d'inodes

2. Le système de fichier FAT a été introduit dans le passé dans le système MSDOS mais reste utilisé dans divers périphériques multimédia actuels. Supposons un système FAT-32 (28 bits d'adressage de blocs disque). Supposons que la taille d'un bloc disque est de 32 kilo octets.

   a) Calculez la taille maximale d'une partition (nous négligeons l'espace disque réservé sur la partition par le système d'exploitation). 1

   Nombre de blocs × taille_bloc = 2^{28} × 32 ko = 2^{28} × 2^5 x 2^{10} = 2^{13} Go

   b) Afin d'accélérer l'accès à la table FAT, le système la charge en mémoire centrale. Si je crée une partition FAT de 32 giga octets, calculez la taille occupée en mémoire par la FAT. 1

   Taille_partition/Taille_bloc)x taille_adresse= (32Go / 32 ko) x 8 octets = 8 Mo