Systèmes d'exploitation

Examen

2023-06-21

Nom et prénom : .....

Numéro d'étudiant : .....

Répondez directement sur le sujet. Si vous manquez de place, vous pouvez écrire au verso.

Le sujet d'examen fait 11 pages. Il comporte 8 exercices pour un total de 22 questions.

Les calculatrices sont interdites.

Exercice 1 : questions générales

1. Quels sont les deux modes de fonctionnement d'un processeur ? Qu'est-ce qui les différencie ? Par quel(s) programme(s) chaque mode est utilisé ? Pourquoi ?

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

2. Classez les types de mémoires qu'on trouve dans un ordinateur du plus rapide au moins rapide.

.....

.....

.....

3. Qu'est-ce que le pointeur d'instruction (*program counter*) ?

.....

.....

.....

.....

4. On considère un fichier `especes.txt` contenant une liste de noms d'espèces animales. Chaque nom d'espèce se trouve sur sa propre ligne. Écrivez une commande en shell Bash permettant d'enregistrer dans un fichier `top10.txt` les 10 premiers éléments de la liste lorsqu'elle est triée par ordre alphabétique.

.....

.....

5. On souhaite implémenter un serveur HTTP. Celui-ci doit pouvoir traiter plusieurs requêtes de façon concurrente. Décrivez dans les grandes lignes une implémentation possible faisant appel à des mécanismes du système d'exploitation.

6. L'erreur de segmentation (*segmentation fault*) est une erreur assez courante pour les programmes en C. Donnez un exemple d'instructions en C susceptibles de causer une erreur de segmentation. Expliquez ensuite pourquoi une telle erreur se produit.

7. En quoi le système d'exploitation a-t-il besoin d'un support du matériel (*hardware*) pour basculer entre les processus dans une approche non-coopérative (préemptive) ?

8. Expliquez la différence entre un lien physique (*hard link*) et un lien symbolique (*soft link*) dans un système de fichiers Unix.

.....

.....

.....

.....

.....

.....

.....

.....

Exercice 2 : ordonnancement

Les politiques d'ordonnancement peuvent facilement être représentées sur un schéma. Par exemple, supposons que l'on exécute la tâche `A` pendant 10 unités de temps, puis que l'on exécute `B` pendant 10 unités également. Notre schéma de cette politique d'ordonnancement aurait l'apparence suivante :

```
    |
CPU | AAAAAAAAAABBBBBBBBBB
    | --------------------
    0          10        20
```

Dans cet exercice, vous devrez montrer votre compréhension des politiques d'ordonnancement en réalisant des schémas similaires.

1. (a) Dessinez un schéma pour la politique d'ordonnancement *Shortest job first (SJF)* avec trois tâches, `A`, `B` et `C`, avec des temps d'exécution respectifs de 15, 10 et 5 unités de temps. `A` et `B` arrivent dans le système à l'instant `0`, `C` arrive à l'instant `4`.

Assurez-vous de compléter l'axe des abcisses de manière appropriée.

```
    |
CPU |
    | ------------------------------------------------------------------------
    0
```

1. (b) Quel est le temps de traitement moyen *(turnaround time)* et le temps de réponse moyen étant donné cet ordonnancement *SJF* des tâches ?

.....

.....

.....

2. (a) Dessinez un schéma pour la politique d'ordonnancement *Round robin (RR)* avec trois tâches, `A`, `B` et `C`, avec des temps d'exécution de 6 unités de temps, en supposant des tranches de temps de taille 2 unités de temps. Les tâches arrivent toutes dans le système à l'instant `0`.

Assurez-vous de compléter l'axe des abcisses de manière appropriée.

```
    |
CPU |
    | ------------------------------------------------------------------------
    0
```

2. (b) Quel est le temps de traitement moyen *(turnaround time)* et le temps de réponse moyen étant donné cet ordonnancement *RR* des tâches ?

.....

.....

.....

Exercice 3 : virtualisation de la mémoire

Dans un système utilisant des registres *base* et *bounds* pour virtualiser un petit espace d'adressage, on a la trace de références mémoire suivante :

```
Virtual Address Trace
    VA  0:  0x000002EE (decimal: 750)  -->  VALID: 0x000036B0 (decimal: 14000)
    VA  1:  0x0000021C (decimal: 540)  -->  VALID: 0x000035DE (decimal: 13790)
    VA  2:  0x000003A8 (decimal: 936)  -->  SEGMENTATION VIOLATION
```

Que peut-on en déduire sur la valeur du registre *base* ? Quelle estimation de la valeur du registre *bounds* peut-on donner ?

.....

.....

.....

.....

.....

.....

.....

.....

Exercice 4 : pagination et TLB

1. Quelles sont les étapes pour accéder à une adresse mémoire avec une gestion de la mémoire par pagination (sans TLB) ? Combien de références mémoires sont nécessaires pour accéder à une adresse mémoire donnée, avec cette approche ?

.....

.....

.....

.....

.....

.....

.....

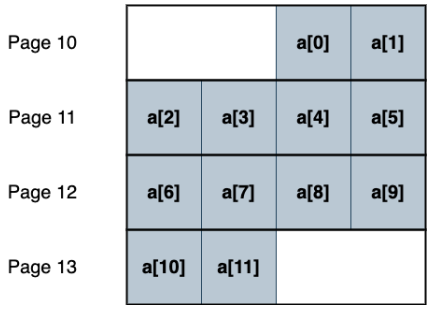
.....

On considère à présent un système avec un TLB ne comportant qu'une seule entrée.

Étant donné le programme suivant :

```
int product = 0;
for (int i=0; i<12; i++) {
    product = product * a[i];
}
```

et la disposition des données dans les pages mémoire suivante :



2. Combien de TLB "*hits*" et "*misses*" se produisent lors de l'exécution du programme ? De quelle propriété du programme le TLB tire profit pour améliorer les performances ?

.....

.....

.....

.....

Exercice 5 : implémentation d'un verrou

On considère la primitive `FetchAndStoreOne()`. Elle exécute une unique instruction atomique et est définie de la façon suivante :

```
int FetchAndStoreOne(int *ptr) {
    int old = *ptr; // récupère l'ancienne valeur dans 'ptr'
    *ptr = 1;       // donne à 'ptr' la valeur 1
    return old;     // renvoie l'ancienne valeur
}
```

Vous devez définir les fonctions `lock_init()`, `lock()` et `unlock()` et une structure `lock_t` pour implémenter un *spin lock* en utilisant `FetchAndStoreOne()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Exercice 6 : problème du à la concurrence

On considère le code suivant qui additionne deux vecteurs et le fait d'une manière sûre pour le multithread.

```
void vector_add(vector *v1, vector *v2) {
    mutex_lock(v1->lock);
    mutex_lock(v2->lock);
    for (i = 0; i < v1->size; i++) {
        v1[i] = v1[i] + v2[i];
    }
    mutex_unlock(v1->lock);
    mutex_unlock(v2->lock);
}
```

On vous dit alors que deux threads différents, 1 et 2, s'exécutent simultanément et appellent ce code de la façon suivante :

```
// Thread 1:                          // Thread 2:
vector_add(&vectorA, &vectorB);       vector_add(&vectorB, &vectorA);
```

Quel problème peut se produire ? Expliquez pourquoi (vous pouvez faire un dessin). Comment peut-on résoudre ce problème ?

.....

.....

.....

.....

.....

Exercice 7 : un autre problème du à la concurrence

On considère à présent le programme suivant.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct __sysinfo {
    char *status
} sysinfo_t;

sysinfo_t *s;

void *init(void *arg) {
    s = (sysinfo_t *) malloc(sizeof(sysinfo_t));
    s->status = "READY";
    return NULL;
}

int main() {
    pthread_t t;
    pthread_create(&t, NULL, init, NULL);

    printf("system status: %s\n", s->status);

    pthread_join(t, NULL);
    free(s);
    return 0;
}
```

Ce programme n'est pas correct. Quel problème peut se produire lors de l'exécution ? Comment rendre ce programme correct ? Vous pouvez annoter le code pour vos explications si nécessaire.

.....

.....

.....

.....

.....

.....

.....

.....

Exercice 8 : système de fichiers et journalisation

Considérons le *very simple file system* que nous avons étudié en classe. Il possède un seul super-bloc, un seul tableau de bits (*bitmap*) de données (DB), un seul tableau de bits (*bitmap*) d'inodes (IB), une série de blocs d'inode (I) et une série de blocs de données (D) ; tous les blocs ont une taille de 4 KB (4096 octets).

1. On suppose que chaque inode possède 6 pointeurs directs et 2 pointeurs indirects. Les adresses de disques sont sur 8 octets. Quelle est la taille maximale d'un fichier dans le système de fichiers ? Indiquez les étapes du calcul, il n'est pas nécessaire de donner le résultat final.

.....

.....

.....

.....

.....

.....

2. Décrivez les opérations qui prennent place dans le système de fichiers lorsqu'on ouvre le fichier `/home/image.txt` avec l'appel système `open("/home/image.txt", ...)`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

3. On suppose que le système de fichiers n'est pas journalisé. On souhaite mettre à jour le bloc de données `D2` dans les blocs de données du fichier déjà existant `/home/image.txt`. Donnez un exemple dans lequel le système de fichiers se retrouve dans un état incohérent suite à une défaillance lors de la mise à jour des structures de données.

.....

.....

.....

.....

.....

.....

4. On suppose à présent que le système de fichiers est journalisé, avec une journalisation physique. On souhaite mettre à jour le bloc de données `D2` dans les blocs de données du fichier `/home/image.txt`. Quel protocole doit-on suivre pour écrire ces nouvelles données tout en garantissant que le système ne se retrouvera pas dans un état incohérent en cas de panne ?

.....

.....

.....

.....

.....

.....

.....