Systèmes d'exploitation

Examen

2023-08-29

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 14 pages. Il comporte 6 exercices pour un total de 23 questions.

Les calculatrices sont interdites.

Exercice 1 : questions générales

1. Quelles sont les deux fonctions principales d'un système d'exploitation ?

2. Quels sont les types de multiplexage possibles ? Quel type de multiplexage peut être utilisé pour partager l'utilisation de la mémoire ? Du processeur ?

3. Dans quel mode de fonctionnement du processeur le shell s'exécute-t-il ?

4. À quoi sert un appel système dans un système d'exploitation ? Donnez un exemple d'appel système et expliquez son fonctionnement.

5. Qu'est-ce qu'une commutation de contexte (*context switch*) ? Donnez deux raisons pour lesquelles une commutation de contexte peut se produire ?

6. On considère le code en C ci-dessous.

```c
int main() {
    fork();
    fork();
    return 0;
}
```

Combien de processus au total sont créés lors de l'exécution de ce programme ?

.....

.....

7. Lorsqu'on souhaite afficher une page web dans son navigateur, celui-ci doit télécharger les éléments de la page depuis un ou plusieurs emplacements distants. Il doit également réaliser un rendu graphique de ces éléments pour les afficher à l'écran. Quel mécanisme du système d'exploitation pourrait-on utiliser pour faire en sorte que l'affichage d'une page web par un navigateur se fasse rapidement ?

.....

.....

.....

.....

.....

.....

.....

.....

8. Écrivez une commande shell qui stocke dans le fichiers `recentes.txt` la liste des 10 dernières commandes shell précédemment exécutées.

.....

.....

9. On considère le programme ci-dessous.

```c
int main(int argc, char *argv[]) {
    printf("%p\n", main);
    printf("%p\n", malloc(100e6));
    int x = 3;
    printf("%p\n", &x);
    return 0;
}
```

Lors de son exécution, on obtient l'affichage suivant dans la console :

```
0x40057d
0xcf2010
0x7fff9ca45fcc
```

Quelle adresse mémoire (virtuelle) se trouve dans la pile (*stack*) et quelle adresse se trouve dans le tas (*heap*) ?

.....

.....

Exercice 2 : multi-level feedback queue

1. Quelles sont les règles d'une politique d'ordonnancement multi-level feedback queue (MLFQ) ?

.....

.....

.....

.....

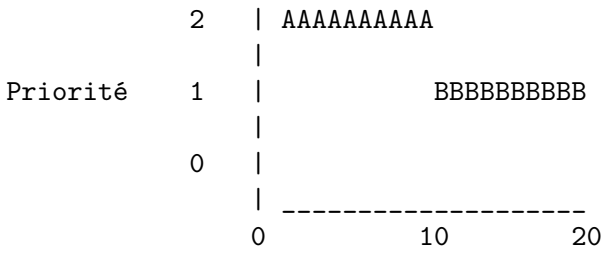
.....

.....

.....

On considère un ordonnanceur multi-level feedback queue (MLFQ). Dans cette question, vous devez dessiner un schéma pour indiquer comment l'exécution des tâches se déroule avec cet ordonnanceur.

Par exemple, supposons que l'on exécute la tâche `A` pendant 10 unités de temps, au niveau de priorité 2, puis que l'on exécute `B` pendant 10 unités au niveau de priorité 1. Notre schéma aurait l'apparence suivante :



Ainsi, l'axe des ordonnées indique la priorité de la tâche en cours d'exécution. Notez bien que seul la tâche en cours d'exécution apparaît sur le schéma, les tâches en attente dans les files n'apparaissent pas.

2. On considère un MLFQ à 3 niveaux (la priorité la plus haute est 2, la plus basse 0). L'ordonnanceur ne déplace pas les tâches entre les files d'attente. On considère deux tâches (`A` et `B`). Ces tâches ne réalisent pas d'entrées/sorties. Elles ont des temps d'exécution de 10 unités de temps et arrivent dans le système à `T=0`. La tranche de temps d'exécution (sans interruption) est de 1 unité de temps au niveau de priorité le plus haut, 2 au niveau moyen et 3 au niveau le plus bas.

Dessinez un schéma pour indiquer comment ces tâches seront executées par l'ordonnanceur. Assurez-vous de compléter l'axe des abscisses de manière appropriée.

3. (a) On considère à présent un scénario dans lequel les paramètres de l'ordonnanceur sont les mêmes que dans la question 2. Les tâches sont différentes : `A` et `B` s'exécutent pendant 10 unités de temps et ne réalisent pas d'entrées/sorties. En revanche, cette fois, `A` arrive dans le système à `T=0` et B à `T=6`.

Dessinez un schéma pour indiquer comment ces tâches seront executées par l'ordonnanceur. Assurez-vous de compléter l'axe des abscisses de manière appropriée.

3. (b) Étant donné l'ordonnancement des tâches dans 3. (a), quel est le temps de traitement *(turnaround time)* et le temps de réponse de la tâche A ?

.....

.....

.....

3. (c) Étant donné l'ordonnancement des tâches dans 3. (a), quel est le temps de traitement *(turnaround time)* et le temps de réponse de la tâche B ?

.....

.....

.....

Exercice 3 : translation-lookaside buffer (TLB)

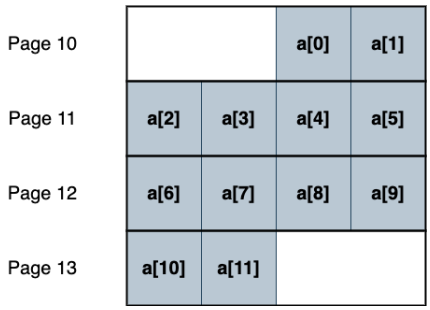
On considère un système avec un TLB comportant deux entrées.

Étant donné le programme suivant :

```
int product = 1;

int i = 0;
for (int j=0; j < 6; j++) {
    product = product * a[i];
    i = (i + 4) % 12;                // rappel: % est l'opérateur modulo
}
```

et la disposition des données dans les pages mémoire suivante :



1. Combien de TLB "*hits*" et "*misses*" se produisent avec une politique de remplacement des entrées *least recently used (LRU)*, lors de l'exécution du programme ? Expliquez.

.....

.....

.....

.....

.....

2. Quelle politique de remplacement des entrées privilégier dans ce cas là ? Combien de TLB "*hits*" et "*misses*" peut on espérer dans le meilleur des cas ? Expliquez.

.....

.....

.....

.....

.....

.....

.....

Exercice 4 : implémentation d'un verrou

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

```
int CompareAndSwap(int *ptr, int expected, int new) {
    int actual = *ptr;
    if (actual == expected) { *ptr = new; }
    return actual;
}
```

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

Exercice 5 : primitives de synchronisation

Dans cet exercice, on vous présente le code source de deux programmes *multi-threads*. Ces programmes ne sont pas corrects : suivant l'ordre d'exécution des threads on peut obtenir des résultats inattendus.

Annotez le code de ces programmes avec les primitives de synchronisation adaptées pour rendre ces programmes corrects et faire en sorte qu'ils produisent des résultats déterminés. La documentation sur les primitives de synchronisation est fournie en annexe.

1.

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



int compte = 100;

void *retrait(void *args) {


    compte = compte - 10;


    return NULL;
}

void *depot(void *args) {


    compte = compte + 20;


    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, retrait, NULL);
    pthread_create(&t2, NULL, depot, NULL);


    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("Montant sur le compte: %d\n", compte);
    return 0;
}
```

2.

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



int continuer = 0;

void *thread1(void *args) {


    printf("Ce message doit s'afficher en premier!\n");




    return NULL;
}

void *thread2(void *args) {







    printf("Ce message doit s'afficher en second!\n");


    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);


    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
```

Exercice 6 : 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).

1. Comment un inode fait-il référence aux blocs de données d'un fichier ?

.....

.....

.....

.....

.....

.....

2. Un nouveau fichier `/home/new.txt` a été ouvert grâce à l'appel système `open()` et un descripteur de fichier a été obtenu. Décrivez les opérations qui prennent place dans le système de fichiers lorsqu'on écrit dans le fichier `/home/new.txt` avec l'appel système `write()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

3. On suppose que le système de fichiers n'est pas journalisé. On souhaite allouer le bloc de données `D2` pour le fichier `/home/new.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 du tableau de bits de données (DB).

.....

.....

.....

.....

.....

.....

4. On suppose à présent que le système de fichiers est journalisé, avec une journalisation des métadonnées. On souhaite mettre à jour le bloc de données `D2` dans les blocs de données du fichier `/home/new.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 ?

.....

.....

.....

.....

.....

.....

.....

Annexes

pthread_mutex_lock(3p) — Linux manual page

NAME

```
pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock - lock and unlock a mutex
```

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

DESCRIPTION

The mutex object referenced by mutex shall be locked by a call to pthread_mutex_lock() that returns zero or [EOWNERDEAD]. If the mutex is already locked by another thread, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by mutex in the locked state with the calling thread as its owner.

pthread_mutex_destroy(3p) — Linux manual page

NAME

```
pthread_mutex_destroy, pthread_mutex_init - destroy and initialize a mutex
```

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
    const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

DESCRIPTION

[...]

In cases where default mutex attributes are appropriate, the macro PTHREAD_MUTEX_INITIALIZER can be used to initialize mutexes. The effect shall be equivalent to dynamic initialization by a call to pthread_mutex_init() with parameter attr specified as NULL, except that no error checks are performed.

pthread_cond_timedwait(3p) — Linux manual page

NAME

pthread_cond_timedwait, pthread_cond_wait - wait on a condition

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_timedwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
int pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
```

DESCRIPTION

The pthread_cond_timedwait() and pthread_cond_wait() functions shall block on a condition variable. The application shall ensure that these functions are called with mutex locked by the calling thread; otherwise, an error (for PTHREAD_MUTEX_ERRORCHECK and robust mutexes) or undefined behavior (for other mutexes) results.

pthread_cond_destroy(3p) — Linux manual page

NAME

pthread_cond_destroy, pthread_cond_init - destroy and initialize condition variables

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_cond_init(pthread_cond_t *restrict cond,
    const pthread_condattr_t *restrict attr);
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

DESCRIPTION

[...]

In cases where default condition variable attributes are appropriate, the macro PTHREAD_COND_INITIALIZER can be used to initialize condition variables. The effect shall be equivalent to dynamic initialization by a call to pthread_cond_init() with parameter attr specified as NULL, except that no error checks are performed.