

Maîtrise d'informatique — Systèmes d'exploitation

Devoir Surveillé

Durée : 2h — Notes de cours autorisées

1 Appels système (exercice d'échauffement)

Question 1 Rappelez brièvement le principe de fonctionnement d'un "appel système" (sur un exemple concret de votre choix). Dans un système d'exploitation multi-utilisateurs, les interactions entre les applications et le noyau s'effectuent principalement au moyen de ce mécanisme. Pourquoi n'est-il pas envisageable d'utiliser de simples appels procéduraux? Même en utilisant des morceaux de code écrits en langage machine, pourquoi une application ne peut-elle contourner ce mécanisme, par exemple pour accéder directement aux structures du noyau?

Question 2 Certains appels systèmes sont destinés à créer (ou à allouer) de nouveaux objets dans le noyau (*e.g.* sémaphores, descripteurs de fichiers) et ils retournent un "identifiant" d'objet (souvent de type opaque) qui sera manipulé par l'application. Pour accélérer les accès ultérieurs à un objet, il serait possible de retourner véritablement un pointeur sur l'objet "déguisé" en type opaque à l'application. Expliquez pourquoi cela n'est pas concevable. Quelle est la bonne manière de faire?

2 Processus légers

Question 1 Rappelez les principaux avantages et inconvénients d'un ordonnanceur de processus légers opérant en espace utilisateur. Même question pour un ordonnanceur de niveau Noyau.

Question 2 Expliquez le principe de l'ordonnancement "hybride" de threads (*i.e.* à deux niveaux comme dans le système Solaris). Que se passe-t-il lorsqu'un thread de niveau utilisateur effectue un appel système bloquant?

Question 3 Qu'est-ce qu'un code réentrant? De quelle manière peut-on transformer un code non-réentrant en code réentrant? Est-ce toujours facile? Pourquoi?

Question 4 En cours, nous avons vu que le noyau Linux gérait **une seule file globale** pour mémoriser la liste des *threads* prêts. Discutez des avantages et inconvénients de cette approche par rapport à une file par processeur.

3 Gestion Mémoire

On considère un système de pagination à deux niveaux :

- les adresses (virtuelles et physiques) sont codées sur 32 bits;
- les 10 premiers bits d'une adresse virtuelle forment le premier index, les 10 suivants forment le second index, et les 12 bits restant forment le déplacement;
- on suppose que chacune des entrées de ces tables occupe 32 bits.

Question 1 Faites un petit dessin illustrant la traduction d'une adresse virtuelle dans ce contexte.

Question 2 Combien d'espace mémoire utilisera-t-on pour gérer la table des pages d'un processus (discuter en fonction de sa taille, etc.)? Dans le pire des cas, quel est le ratio entre la taille occupée par la table et les données du processus? Est-ce une situation réaliste? Pourquoi?

4 Sémaphores construits avec des moniteurs

Dans cet exercice, on se propose d'implémenter des sémaphores en utilisant les mécanismes associés aux moniteurs de Hoare.

```
// Outils disponibles
typedef ??? lock;
typedef ??? condition;
```

```
void lock_init(lock *l);
void lock_acquire(lock *l);
void lock_release(lock *l);
void cond_init(condition *c);
void cond_wait(condition *c, lock *l);
void cond_signal(condition *c, lock *l);
```

```
// Outils à implémenter
typedef ??? semaphore;
```

```
void sem_init(semaphore *s, int value);
void P(semaphore *s);
void V(semaphore *s);
```

Question En supposant définis les types `lock` et `condition` ainsi que les opérations associées (voir figure), définissez le type `semaphore` et donnez le code des fonctions `sem_init`, `P` et `V`.

5 Nachos et les sémaphores

Si l'on regarde la façon dont sont implantés les **sémaphores** dans Nachos, on remarque la présence curieuse d'une boucle dans la méthode `P()` (la fonction `V()` est donnée pour information) :

```
void Semaphore::P()
{
    IntStatus oldLevel = interrupt->SetLevel(IntOff); // disable interrupts

    while (value == 0) { // semaphore not available
        queue->Append((void *)currentThread); // so go to sleep
        currentThread->Sleep();
    }
    value--; // semaphore available,
            // consume its value

    (void) interrupt->SetLevel(oldLevel); // re-enable interrupts
}

void Semaphore::V()
{
    Thread *thread;
    IntStatus oldLevel = interrupt->SetLevel(IntOff);

    thread = (Thread *)queue->Remove();
    if (thread != NULL) // make thread ready, consuming the V immediately
        scheduler->ReadyToRun(thread);
    value++;
    (void) interrupt->SetLevel(oldLevel);
}
```

Question 1 Selon vous, pour quelle raison n'utilise-t-on pas un simple test (i.e. `if`) au lieu d'une boucle `while` ?

Question 2 En ne faisant aucune hypothèse particulière sur l'algorithme d'ordonnancement de Nachos (c.-à-d. que l'on ignore la façon dont fonctionne `findNextToRun`), y-a-t'il un risque d'aboutir à des situations de famine si on utilise par exemple les sémaphores pour implanter des sections critiques ?