université de BORDEAUX

Collège Sciences et technologies

Année	2021-2022	**Type**	Devoir Surveillé
Master	Informatique		
Code UE	4TIN705U	**Épreuve**	Systèmes d'Exploitation
Date	13/11/2021	**Documents**	Non autorisés
Début	10h30	**Durée**	1h30

Correction du DS

1 Questions de cours (échauffement)

Question 1 La segmentation mémoire permet de diviser l'espace d'adressage d'un processus en plusieurs segments (par exemple le segment de code, le segment de pile, etc.) et permet au système d'exploitation de placer ces segments en mémoire physique de manière indépendante. Il est ainsi plus facile de charger de nouveaux processus en mémoire lorsque celle-ci est fragmentée.

Pour mettre en œeuvre la segmentation mémoire, on repose sur un processeur capable de convertir les adresses virtuelles (identifiant de segment + déplacement au sein du segment) en adresse physique en utilisant des registres spéciaux pour chaque segment (si le nombre de segments est réduit) ou une table en mémoire (si le nombre de segments est grand).

Pour le joli dessin, voir par exemple celui de la page 49 du diaporama du cours. Les avantages/inconvénients sont dressés quelques diapos plus loin.

Question 2 Un thread est un flot d'exécution indépendant, pour lequel le noyau alloue un bloc de contrôle (identifiant, priorité, état, espace pour sauver le contexte) ainsi qu'une pile. Tous les noyaux modernes ordonnancent des threads.

Un processus est composé d'un espace d'adressage (code, données, tas, etc.) et d'un ou plusieurs threads qui s'exécutent à l'intérieur. Tous les threads d'un même processus partagent donc un espace d'adressage identique.

Le noyau ordonnance avant tout des threads. Toutefois, le noyau peut tenir compte du fait que deux threads appartiennent à un même espace d'adressage pour favoriser un changement de contexte de l'un vers l'autre (et ainsi éviter de reconfigurer la MMU).

Voir « *The Big Picture* » dans le diaporama du cours, page 104.

Question 3 TODO.

2 Nachos

Question 1 Lors d'un appel à `Semaphore::V()`, on voit dans le code que l'on se contente de réveiller un thread qui devra potentiellement lutter avec d'autres threads pour consommer le jeton. Il est donc possible qu'il se fasse doubler par un thread qui appelle `Semaphore::P()` pour la première fois. D'où la nécessité d'utiliser une boucle `while`...

Question 2 Lors de l'appel `currentThread->Sleep()`, il se produit un changement de contexte vers un thread qui avait précédemment lui aussi cédé la main (dans la fonction `Sleep`, `Yield`, etc.). Ce qui importe est que ce thread se réveille au beau milieu d'une fonction qui se termine également par `interrupt->SetLevel (oldLevel)`.

Question 3

```
void Mutex::Lock ()
{
    IntStatus oldLevel = interrupt->SetLevel (IntOff);

    while (owner != NULL) {
          queue->Append ((void *) currentThread);
          currentThread->Sleep ();
    }
    owner = currentThread;

    (void) interrupt->SetLevel (oldLevel);
}

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

    ASSERT(owner == currentThread);

    thread = (Thread *) queue->Remove ();
    if (thread != NULL)
        scheduler->ReadyToRun (thread);
    owner = NULL;
    (void) interrupt->SetLevel (oldLevel);
}
```

Question 4

```
void Condition::Wait (Mutex *m)
{
    IntStatus oldLevel = interrupt->SetLevel (IntOff);

    m->Unlock ();

    queue->Append ((void *) currentThread);
    currentThread->Sleep ();

    (void) interrupt->SetLevel (oldLevel);

    m->Lock ();
}

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

    thread = (Thread *) queue->Remove ();
    if (thread != NULL)
        scheduler->ReadyToRun (thread);

    (void) interrupt->SetLevel (oldLevel);
}
```

3 Barrières de synchronisation en deux temps

Note : on ne demandait pas de rendre ces primitives réutilisables plusieurs fois de suite...

Question 1

```
Semaphore mutex(1), wait(0);
int nb = 0;

void signaler ()
{
  P(mutex);
  if (++nb == MAX)
    while (nb--)
      V(wait); // on distribue nb jetons
  V(mutex);
}

void attendre ()
{
  P(wait);
}
```

Question 2

```
mutex m;
cond c;
int nb = 0;

void signaler ()
{
  mutex_lock (m);
  if (++nb == MAX)
    cond_bcast(c);
  mutex_unlock(m);
}

void attendre ()
{
  mutex_lock (m);
  while (nb < MAX)
    cond_wait(c, m);
  mutex_unlock(m);
}
```