NACHOS : Kernel threads

Etape 2+
Polytech RICM2

2004–2005

25 janvier 2005

L'objectif de cette etape est de modifier l'ordonnancement des threads noyaux et de compléter l'implantation de mécanismes de synchronisation .

Partie I. Etude des threads noyau

Vous allez examiner en détail le fonctionnement des threads noyau Nachos.

Action I.1. *Comment ces threads sont-ils alloués et initialisés ? Où se trouve la pile d'un thread Nachos, en tant que thread* noyau *? Décrire ce qui se passe au démarrage d'un thread et à sa fin ?*

Action I.2. *Où sont mis les descripteurs des threads prêts à s'exécuter ? Comment sont ils gérés ? Que font les méthodes* `Sleep` *et* `Yield` *?*

Action I.3. *Où sont mis les descripteurs des threads prêts à s'exécuter ? Comment sont ils gérés ? Que font les méthodes* `Sleep` *et* `Yield` *?*

Action I.4. *Comment sont implantés les* `Semaphore` *de Nachos ?*

Partie II. Temps partagé

Il est possible de faire exécuter Nachos avec une commutation des threads sur des intervalles de temps calculés aléatoirement.

```
nachos -rs valeur_du_germe_aléatoire
```

Action II.1. *Que se passe-t-il au démarrage de Nachos (* `Initialize` *dans* `system.cc` *) ? Quelle procédure est appelée sur une interruption horloge ? et que fait elle ?*

Action II.2. *Modifier le programme* `threadtest` *:*

```
Semaphore * mutex;

void SimpleThread (int which)
{int num;
    for (num = 0; num < 10; num++)
      {   mutex->P();
          printf ("*** thread %d looped %d times\n", which, num);
  mutex->V();
      }
}

void ThreadTest ()
```

```
{Thread *t ;
 mutex=new Semaphore("MUTEX",1);
    DEBUG ('t', "Entering SimpleTest\n");

    for(int j=1;j<=10;j++){
        t= new Thread ("forked thread");
        t->Fork (SimpleThread, j);
}
    SimpleThread (0);
}
```

Faites le exécuter sans partage de temps, avec partage de temps. Regarder[1] ce qui se passe avec un même germe et différents germes ? Que pouvez dire sur les ordres d'exécution de threads ainsi obtenus ?

Partie III. Ordonnancement noyau équitable

Action III.1. *Modifier le lancement de nachos (Voir `Initialize` dans `system.cc`)pour démarrer le temps partagé avec un quantum fixe. Pourquoi dit-on que la gestion du processeur est équitable ?*

Action III.2. *L'implantation des sémaphores peut être source de famine pour un thread . Pourquoi ? Pouvez vous faire un test le mettant en évidence ? Corriger et tester ?*

Partie IV. Implantation des moniteurs

Action IV.1. *Proposer une implantation[2] équitable des classes `Lock` et `Condition` implantant une statégie priorité aux réveillés sur les arrivants - priorité au signaleur sur les réveillés.*

Action IV.2. *Ecrire un jeu de test utilisant votre solution : Allocation de ressources Lecteur-Rédacteur, N philisophes, Producteur-consommateur.*

Partie V. BONUS : Extension temps réel

Les threads sont gérés par priorité. L'intervalle des priorité est fixé de MIN_PRIO (0) à MAX_PRIO.

Action V.1. *Proposer une classe ListPrio gérent des listes classées par priorité. Donner un jeu de test la validant.*

Action V.2. *Modifier l'ordonnancement des threads prêts pour garantir que le thread le + prioritaire est actif. Dans un même niveau de priorité, les threads sont gérés en temps partagé. Donner un jeu de tests.*

Action V.3. *Modifier l'implantation des verrous et conditions pour garantir l'attribution du verrou au thread le plus prioritaire. On ignore le phénomène d'inversion de priorité. Donner un jeu de test.*

Partie VI. BONUS++ : Inversion de priorité

Action VI.1. *Proposer un test mettant en évidence le phénomène d'inversion de priorité.*

Action VI.2. *Proposer une solution à ce problème par* **héritage de priorité**. *Donner un jeu de test.*

[1]On peut enregistrer les différentes traces d'exécution `nachos <paramètre> > trac_file_name` et comparer les fichiers de trace par la commande `diff`

[2]Sans utiliser les sémaphores.