Université Paris 7
Master 1 II – Théorie et pratique de la concurrence
14 juin 2006

Durée : 3 heures.
Documents autorisés : notes de cours et TD. Le barème est indicatif.

Question 1 (3 points)

Soit l'algorithme d'exclusion mutuelle distribué de Ricart-Agrawala vu en cours.

1. Quel est le nombre total de messages envoyés dans le cas où tous les processus veulent entrer en section critique ?
2. Construire un scénario où les valeurs des tickets augmentent sans borne.
3. Quelle est la différence maximum entre deux numéros de tickets ?
4. Dans l'algorithme final, que se passe-t-il si plusieurs noeuds ont choisit le même numéro ?

Question 2 (4 points)

Dans cet exercice, on vous demande d'écrire en Promela un algorithme concurrent distribué pour tester si les séquences de feuilles de deux arbres binaires sont égales. Par exemple, les arbres binaires définis par les expressions suivantes $(a,(b,c))$ et $((a,b),c)$ ont la même séquence des feuilles.

On suppose qu'on dispose en Promela d'un type `btree_t` pour représenter les valeurs de type arbre binaire. Pour ce type, les opérations suivantes sont disponibles : `valFeuille` (-1 si pas feuille, valeur feuille sinon), `filsGauche` et `filsDroit`.

L'algorithme est composé de trois processus : deux processus qui parcourent chacun son arbre et envoient les feuilles rencontrées dans l'ordre au troisième processus pour les comparer.

Question 3 (5 points)

Une tribu de sauvages mange un dîner commun préparé dans une grande marmite qui peut contenir au plus M morceaux. Quand un sauvage désire manger, il se sert directement dans la marmite sauf si elle est vide. Si la marmite est vide, le sauvage réveille le cuisinier et attend que le cuisinier remplisse la marmite avant de se servir. Les sauvages ne font que se servir et manger ; le cuisinier ne fait que dormir et remplir de M morceaux la marmite. Manger et remplir la marmite prennent du temps mais elles se terminent forcement.

Donner une solution à ce problème en utilisant comme mécanisme de synchronisation les moniteurs. Ainsi, pour manger, les processus “sauvages” doivent appeler une méthode du moniteur. Le processus “cuisinier” exécute également une méthode du moniteur.

Votre solution doit (1) être écrite en Java richement commenté et (2) indiquer l'invariant du moniteur par rapport aux champs utilisés.

Question 4 (4 points)

Soit l'algorithme d'exclusion mutuelle suivant :

```
#define N 5
int requete = 0, reponse = 0;
active [N] proctype Client() {
  do :: true -> /* section non critique */
    do :: reponse != _pid -> requete = _pid
       :: else -> break
    od;
    /* section critique */
    reponse = 0
  od
}
active proctype Serveur() {
  do :: true -> (requete != 0);
                reponse = requete;
                (reponse == 0);
                requete = 0
  od
}
```

1. Expliquer (10 lignes maximum) comment cet algorithme fonctionne. Rappel : `_pid` est le numéro d'identification du processus dans le tableau de processus.

2. Montrer, en raisonnant sur l'algorithme, quelles sont les propriétés (exclusion mutuelle, absence de blocage et absence de famine) satisfaites par l'algorithme.

Question 5 (6 points)

Etant donné un tableau d'entiers `a[N]`, on veut calculer un tableau `sum[N]` où chaque élément `sum[i]` est la somme des `i` premiers éléments de `a`. Un solution sequentielle simple comporte `N` additions :

```
sum[0] = a[0];
for(i = 1; i < N; i++)
  sum[i] = sum[i-1] + a[i];
```

On désire paralléliser cet algorithme pour arriver à un nombre d'additions logarithmique en `N`. Pour cela, on lance `N` processus en parallèle, chacun calculant un élément de `sum`. Premièrement, tous les `sum[i]` sont affectés à `a[i]`. Ensuite, on additionne en parallèle `sum[i-1]` et `sum[i]` pour tout `i` plus grand que 1, c'est-à-dire qu'on additionne les sommes à distance de 1. Puis, on double la distance et on ajoute `sum[i-2]` à `sum[i]`, pour tout `i` plus grand que 2. Si on continue à doubler la distance, après $\lceil log_2 N \rceil$ tours, toutes les sommes partielles sont calculées par tous les processus.

1. On suppose qu'on dispose d'une primitive de synchronisation "barrière" pour le groupe de `N` processus. Cette primitive fonctionne comme suit : un processus `i` qui appelle `barriere(i)` reste bloqué tant qu'il existe des processus du groupe qui n'ont pas executé cette primitive (il ne sont pas encore arrivés à la barrière). Cette primitive est renouvelable, c'est-à-dire qu'elle peut être appelée autant de fois que nécessaire.

   En utilisant cette primitive, écrire en Promela le programme concurrent qui calcule les sommes partielles en $\lceil log_2 N \rceil$ tours (c'est-à-dire $\lceil log_2 N \rceil$ appels dans chaque processus à la primitive `barriere`).

2. On desire maintenant implementer l'algorithme en Linda. Pour cela, à chaque tour, les processus déposent dans l'espace de travail la somme partielle calculée au tour précèdent, puis il lit les informations necessaires au tour courant et fait l'addition correspondante.

   Donner l'implementation en Linda en utilisant les cannaux de Promela.