NOM:_____

Prénom : _____

Examen ARA – 2012
Durée : 2 heures
Documentation autorisée : polycopie du cours
L. Arantes, O. Marin, F. Petit, P. Sens

Exercice: PAXOS

1
Dans l'algorithme de Paxos, pourquoi dans la dernière phase attendre une majorité de réception de message *accept* avant de décider ? Grâce à quel mécanisme l'algorithme assure qu'une majorité d'*accept* est reçue ?

Exercice: Détecteur de fautes

On considère un ensemble de processus $\Pi = \{p_1, p_2, .., p_n\}$ communiquant par messages. Les liens de communication sont bidirectionnels et fiables. On ne considère que des fautes du type « crash ». Le réseau forme un graph complet partiellement synchrone : après le temps GST (inconnu), il est existe des bornes sur les délais de transmissions. Il y a au moins un processus correct.

Considérez l'algorithme suivant :

Every process $p_i, i = 1, \ldots, n$ executes:
$trusted_i \leftarrow 1$
$\forall j \in \{1, \ldots, i-1\} : \Delta_{i,j} \leftarrow$ default timeout
cobegin
|| Task 1: **repeat periodically**
  if $trusted_i = i$ **then** send I-AM-THE-LEADER to $p_{i+1}, \ldots, p_n$
|| Task 2: **when** $(trusted_i < i)$ and
  (did not receive I-AM-THE-LEADER from $p_{trusted_i}$ during the last $\Delta_{i,trusted_i}$ time units)
  $trusted_i \leftarrow trusted_i + 1$
|| Task 3: **when** (received I-AM-THE-LEADER from p_j) and $(j < trusted_i)$
  $trusted_i \leftarrow j$
  $\Delta_{i,j} \leftarrow \Delta_{i,j} + 1$
coend

1

Complétez l'algorithme avec la tâche T4 afin implémenter un détecteur Omega. A terme tout les processus doivent élire le même processus comme leader.

Task T 4 : upon the invocation of *leader ( )*

2

Supposons qu'aucun processus ne tombe en panne. Quel sera le processus leader ? Est-ce que temporairement l'identité d'un autre processus peut être rendu lors de l'appel à *leader ( )* ? Est-ce qu'à un instant t deux processus peuvent rendre l'identité de processus différents lors de l'appel de *leader ( )*? Pourquoi dans la tâche T3 le timeout est incrémenté ? Justifiez vos réponses

On considère que:
correct : ensemble qui contient les processus corrects
pleader : processus correct élu comme leader
leader (t) : invocation de leader à l'instant t.

3

Montrez que :

$$\exists t : \forall t' > t, \forall p_i \in correct, leader(t') = pleader$$

4

Si on ajoute à chaque processus un variable locale *suspected*$_i$ qui est mise à jour à *Π* – {*trusted*$_i$} dans la Task3, est-ce que l'algorithme ci-dessus implémente un détecteur de défaillance <> S ? Justifiez votre réponse.

En s'inspirant sur l'algorithme ci-dessus nous voulons maintenant implémenter un détecteur de défaillance <>P. Les pseudo-codes de l'initialisation des variables et de la Task1 sont les suivants :

Every process $p_i, i = 1, \ldots, n$ executes:
$trusted_i \leftarrow 1$
$suspected_i \leftarrow \emptyset$ $\{suspected_i$ *provides the properties of* $\Diamond P\}$
$\forall j \in \{1, \ldots, n\} : \Delta_{i,j} \leftarrow$ default timeout $\{\Delta_{i,j}, j < i$ *are used to eventually agree on a common leader process*$\}$
$\{\Delta_{i,j}, j > i$ *are used by the leader to build the set of suspected processes*$\}$

cobegin
|| Task 1: repeat periodically
    if $trusted_i = i$ then
        send (I-AM-THE-LEADER, $suspected_i$) to $p_{i+1}, \ldots, p_n$
    else
        send I-AM-ALIVE to $p_{trusted_i}$

Notez que l'ensemble *suspect*$_i$ est ajouté dans le message I-*AM-THE-LEADER*

5
Complétez les autres tâches de l'algorithme. Vous pouvez ajoutez le nombre de tâches que vous voulez. Cependant, la seule tâche qui peut envoyer des messages est la Task1.

Exercice : Modèle de Cohérence

Dans le modèle de cohérence faible(weak), une variable de synchronisation (S) est introduite et utilisée pour synchroniser l'accès à une variable partagée (section critique). Lorsqu'une opération de synchronisation d'un thread (processus) se termine tous les autres voient les dernières écritures faites par le thread qui possédait la variable de synchronisation.

Le modèle de cohérence faible définit les 3 propriétés suivantes :

- Les accès à des variables de synchronisation respectent la cohérence séquentielle.
- Aucun accès à des variables de synchronisation n'est permis jusqu'à ce que tous les threads complètent tous les opérations d'écritures effectuées précédemment.
- Aucun accès à une donnée partagée (lecture ou écriture) n'est permis jusqu'à ce que tous accès précédents à des variables partagées aient été complétés.

1
Sachant que les accès aux variables partagées respectent la cohérence séquentielle, quel est l'impact sur l'ordre de ces accès pour les processus ?

Considérez les deux scénarios ci-dessous :

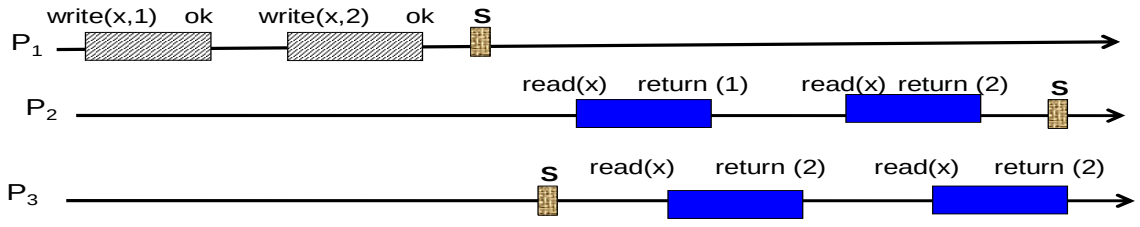


Figure 1 (a) – Scénario 1

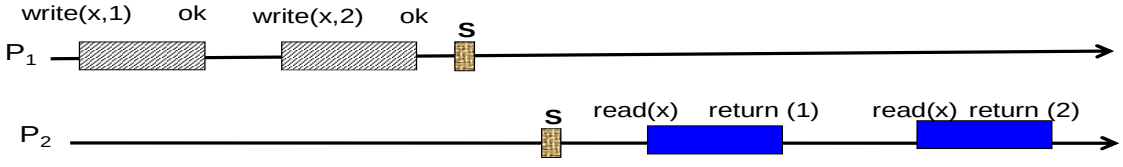
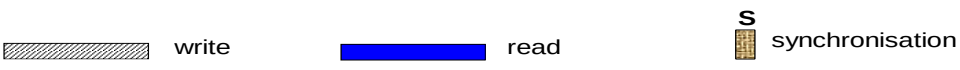


Figure 1 (b) – Scénario 2



2

Est-ce que le scénario 1 respecte le modèle de cohérence faible ? Et le scénario 2 ? Justifiez vos réponses.