M2 – SAR – 2008

Luciana Arantes, Maria Gradinariu, Pierre Sens

ARA
Durée : 3 heures
Documentation autorisée : polycopie

Exercice 1: Détecteur de fautes sans temporisateur

On considère, un ensemble de processus $\Pi = \{p_1, p_2, .., p_n\}$ communiquant par messages. Les liens de communication forment un graphe complet (chaque processus peut directement envoyer un message à chacun des processus). Les liens de communication sont fiables.

On considère des fautes de type « crash » avec ***f*** le nombre maximum de fautes.

Soit le détecteur de fautes associé à p_i dont le code est le suivant

1. **Initialisation**
2. $suspected_i = \emptyset$

3. **Tâche 1 :**
4. Boucle infinie
5. Pour k = 1 à n
6. Envoyer REQUEST à p_k
7. Attendre la réception de *n – f* message *RESPONSE*
8. $recv\_from_i$ = L'ensemble de processus qui ont répondu à la dernière requête
9. $suspected_i = \Pi - recv\_from_i$
10. Fin boucle

11. **Tâche 2 :** Réception de *REQUEST* de p_j
12. Envoyer *RESPONSE* à p_j

1.1

Pourquoi à la ligne 7 attend on ***n-f*** et non pas **n** messages ?

On souhaite dans la suite que cet algorithme implémente un détecteur **<> S**.

1.2

Quelles hypothèses faut il faire sur les canaux pour que ce détecteur assure la propriété de justesse de <>S.

1.3

Indiquez en quelques lignes, pourquoi la complétude est assurée

1.4

Pourquoi cet algorithme entraîne –t –il beaucoup de fausses suspicions ?

1.5

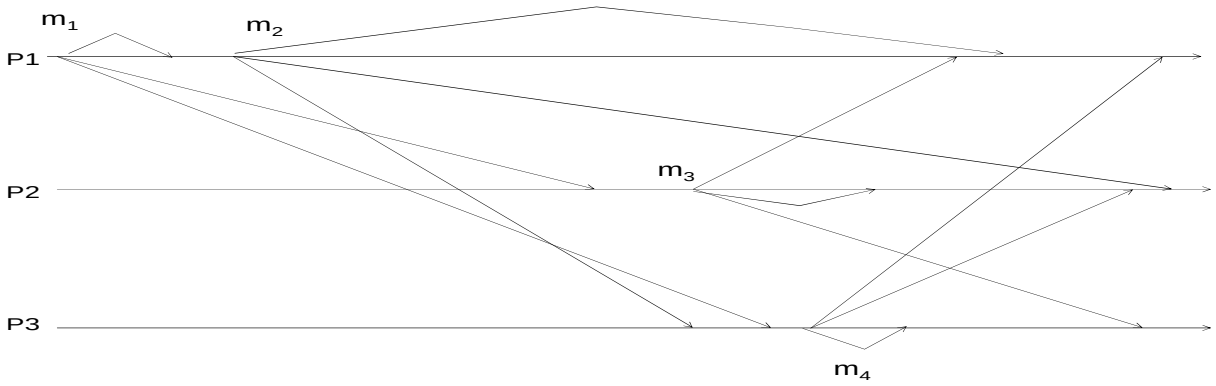
Modifiez l'algorithme pour réduire le nombre de fausses suspicions. (Indication : l'idée est de suspecter les processus qui n'ont répondu à personne).

1.6

Comment faudrait il modifier votre algorithme pour obtenir un détecteur **Ω** (toujours sans utilisé de temporisateur) ? Donnez juste le principe.

Exercice 2 : Protocole de Diffusion Atomique et Causal

Considérez le diagramme de temps ci-dessous :



3.1

Modifiez les délivrances des messages $m_1,\ldots m_4$ pour que leur diffusions soient en même temps causal et atomique. Vous ne pouvez retarder que les délivrances.

Exercice 3 : Protocole de Diffusion Fiable

Considérez l'algorithme de diffusion fiable vu en cours. On veut le modifier afin de réduire au maximum le nombre de retransmission. Pour cela, on supposera l'existence d'un détecteur de défaillance **parfait P** sur chaque processus *p*. Si un processus *q* tombe en panne (pannes franches), *p* est à terme informé et il n'y a pas de fausses suspicions:

Processus p :

upon <crash, q>
  $correct_p = correct_p \setminus \{q\}$

Les canaux sont faibles et les processus sont susceptibles de subir de pannes franches. Le système possède au début *N* processus corrects : $\Pi = \{p_1,p_2,..p_n\}$

3.1

Complétez le corps de la primitive **Real_broadcast (m)** et le code lorsqu'un processus reçoit un message (**upon event** recv(m, p_j)) afin d'offrir une diffusion fiable. Vous pouvez ajouter d'autres variables locales si vous le jugez nécessaire ainsi que d'autres fonctions et/ou événements ou même changer le code du événement <crash,p_j>.

Processus p_i :

Variables locales :

$correct_{pi} = \Pi$; /* Π = ensemble de tous les processus du système */

. . .

Real_broadcast (m)

. . .

upon event recv(m, p_j) **do**

. . .

Real_deliver(m) /* délivrer le message */

upon <crash, p_j>

$correct_{pi} = correct_{pi} \setminus \{ p_j \}$

3.2

Quelle est la complexité en terme de message du nouvel algorithme ? Présente-t-il des inconvénients par rapport à l'algorithme vu en cours ? Justifiez votre réponse.

3.3

Au lieu d'un détecteur P, serait-il possible d'utiliser un détecteur de la classe <> P? Justifiez votre réponse.

Exercice 4 : Robots : Nomage Unique

Considérons un réseau dont la topologie est non-dirigée et sans cycles. Les noeuds du réseau sont anonymes mais pour chaque noeud, *p*, les liens avec ses voisins sont numérotés de 0 à $degre_{p-1}$ où $degre_p$ est le degré du noeud *p* dans le réseau.

Dans ce réseau *n* agents ou robots se déplacent en passant d'un noeud à un de ses voisins. Pour chaque noeud visité les agents connaissent le numéro du lien sur lequel ils y sont entrés. Le lien de sortie sera choisi en fonction d'un calcul que les agents vont faire à partir des informations retrouvées localement sur les noeuds.

Chaque noeud du réseau dispose d'une mémoire appelée par la suite tableau blanc dans laquelle les agents (une fois arrivés sur le noeud) peuvent inscrire des informations (ex. leur identifiant, le lien d'entrée et éventuellement le lien de sortie). Bien évidement les agents ont la possibilité (une fois arrivés sur un noeud) de lire l'état courant du tableau blanc.

En sachant qu'initialement les agents peuvent ne pas se trouver tous sur le même noeud et qu'ils n'ont pas la possibilité de communiquer entre eux mise à part l'échange d'informations via les tableaux blancs, on souhaite mettre en place un nomage unique des agents (au bout d'un temps fini tous les agents doivent avoir un identifiant unique et leur l'identifiant ne changera plus dans le reste de l'exécution du système). Noter qu'initialement les agents peuvent avoir le même identifiant.

L'idée de l'algorithme est simple. Chaque agent commence à parcourir le réseau afin d'inscrire son identité dans tous les tableaux du réseau. La politique de parcours est la suivante: si l'agent est entré dans un noeud, *p*, par le lien *i*, alors il va sortir par le lien $i+1 \bmod degree_p$. A l'arrivée sur un noeud l'agent lit les informations du tableau blanc, décide s'il doit changer d'identifiant ou non puis continue son trajet en marquant dans le tableau son identité (si elle n'existait pas déjà), son lien d'entrée et son lien de sortie. Si au démarrage de l'algorithme l'agent se trouve déjà sur le noeud *p* alors il marque $\perp$ pour le lien d'entrée et *0* pour le lien de sortie.

Par la suite nous allons faire les hypothèses suivantes :

- les tableaux blancs sont tous de taille *n* (*n* étant le nombre d'agents dans le réseau);
- quand deux agents se trouvent sur le même noeud alors il vont être activés un après l'autre ;
- deux agents ne peuvent pas traverser un lien en même temps ;
- la seule information qu'un agent peut stocker est son identifiant.

4.1

Décrire en pseudo-code la procédure de nomage.

4.2

Prouver la correction de votre algorithme (Indication : Montrer qu'au bout d'un temps fini tous les agents auront un identifiant unique et que leurs identifiants ne changeront plus).

4.3

Considérons maintenant une topologie en anneau. Est-ce que le problème de nomage sous les hypothèses considérées a une solution déterministe ? Justifier votre réponse.