AR (MI048) - Écrit réparti - Première épreuve

Les deux exercices doivent être traités sur des copies séparées.

Exercice(s)

Exercice 1 – Horloges et Causalité

Nous nous intéressons dans cet exercice à l'implémentation d'un "*broadcast causal*". Dans ce mécanisme, l'ordre dans lequel les messages sont délivrés sur un site (i.e., transmis à la couche application, ce qui est indépendant de l'ordre dans lequel ils arrivent sur le site) doit respecter la relation de causalité entre les émissions. Autrement dit,

$$\forall m, m' \textit{ deux messages}, \ \textit{broadcast}(m) \rightarrow \textit{broadcast}(m') \ \Rightarrow \ \textit{sur chaque site}, \ \textit{deliver}(m) \rightarrow \textit{deliver}(m')$$

Nous nous plaçons pour cet exercice dans un cadre général : les liaisons sont fiables mais pas nécessairement FIFO.

Question 1

Tous les messages sont-ils forcément délivrés dans le même ordre sur chacun des sites ? Justifiez.

Question 2

On considère l'exécution représentée dans la Figure 1. Quelle est la relation de causalité entre les émissions de messages ? Dans quel ordre les messages sont-ils délivrés sur chacun des sites ? Justifiez.

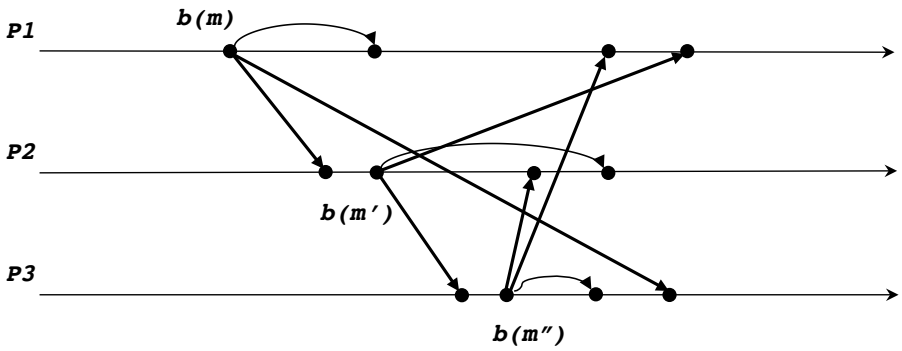

FIGURE 1 – Exécution d'une application avec broadcast

On souhaite mettre en place un mécanisme d'horloges simple (seuls les événements de type *broadcast* sont datés) pour garantir que chaque site délivre les messages dans un ordre respectant la causalité.

Soit $Sites$ l'ensemble des sites de l'application et N le cardinal de cet ensemble. Chaque site i gère un vecteur d'horloges VC_i de taille N, initialisé à 0. $VC_i[x], x \neq i$ doit comptabiliser le nombre de broadcasts effectués par le site x et déjà reçus par i.

On note $m.VC$ l'horloge associée au message m. Lorsque le site i diffuse un message, il exécute la primitive suivante :

Broadcast(m)
$m.VC = VC_i$
$\forall x \in Sites,\ \ envoyer\ m\ à\ x$
$VC_i[i] = VC_i[i] + 1$

Question 3

Lorsque i reçoit un message diffusé par j, quelle condition lui permet de s'assurer qu'il a reçu tous les messages précédemment diffusés par j (précédence locale) ? Vous pourrez vous aider de l'exécution de la Figure 2.

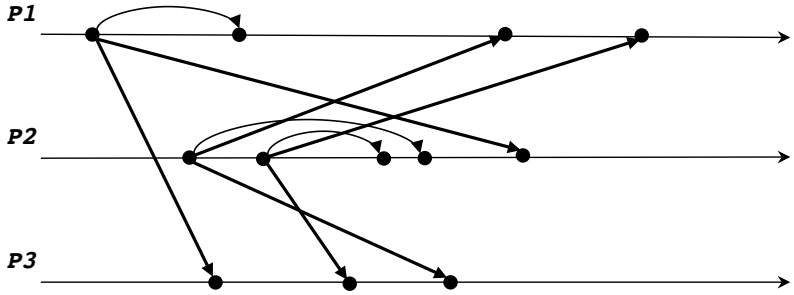

FIGURE 2 – Causal broadcast avec horloges scalaires

Question 4

Lorsque i reçoit un message m diffusé par j, quelle condition lui permet de s'assurer qu'il a reçu tous les messages qui précèdent causalement la diffusion de m ? Justifiez. Déduisez-en la condition à laquelle i peut délivrer m

Question 5

Quel problème poserait l'utilisation d'horloges scalaires ? Justifiez votre réponse. Vous pourrez pour cela vous appuyer sur l'exécution de la Figure 2.

Exercice 2 – Exclusion mutuelle- Algorithme à jeton de Naimi-Tréhel

Considérez l'algorithme d'exclusion mutuelle de Naimi-Tréhel à base de jeton vu en cours. Nous voulons le modifier de la façon suivante :

- La file de *next* n'existe plus.
- La fonction $Request\_CS(S_i)$ ne change pas, ni l'initialisation (à l'exception de la variable $next$ qui n'existe plus).
- Comme dans l'algorithme original, si un site i, non demandeur de la section critique, reçoit un message $REQUEST$ soit i le renvoie à son père, ($father_i \neq NIL$) soit i envoi le jeton au site demandeur j ($father_i = NIL$). Dans ces deux cas, $father_i$ pointera vers j. Cependant, dans cette nouvelle version de l'algorithme, si i est demandeur de la section critique et reçoit un message $REQUEST$, i sauvegarde la requête (requête en attente).
- Fonction $Release\_CS(S_i)$: lorsque le site i libère la section critique, si sa liste de requêtes en attente n'est pas vide, il envoie le jeton à j, un des sites émetteur d'une de ces requêtes. De plus, i envoie à j sa liste des requêtes en attente et met à jour $father_i$.

Question 1

Comment la liste des requêtes en attente d'un site i est-elle organisée ? Si cette liste n'est pas vide, à quel site i enverra-t-il le jeton et sa liste de requêtes en attente ? Justifiez vos réponses.

Question 2

Supposons que j ait une liste de requêtes en attente et reçoive de i la liste de requêtes en attente de celui-ci. Comment j organisera-t-il les requêtes ? Justifiez votre réponse.

Question 3

En considérant le principe de l'algorithme original de Naimi-Thréhel pour l'arbre de $father$, comment le site i devra-t-il mettre à jour sa variable $father_i$ lors de l'exécution de la fonction $Release\_CS(S_i)$? Justifiez votre réponse.

Question 4

Donnez le pseudo-code des fonctions $Release\_CS(S_i)$, $Receive\_REQUEST(S_j)$ et $Receive\_Token(S_j)$. Vous spécifierez les nouvelles variables éventuellement utilisées par l'algorithme. Vous pouvez changer le prototype des fonctions ou le type des variables, si nécessaire. Le pseudo-code de la fonction $Resquest\_CS(S_i)$ et l'initialisation sont donnés dans la Figure 3.

```
Local Variables:
Token : boolean;
requesting; boolean
int father;
```

```
Initialisation de Si:
  father = S1;
  requesting = false;
  Token = (father == Si);
  if (father == Si)
    father = nil;
```

```
Request_CS (Si):
  requesting = true;
  if (father <> 0) {
    send (Request, Si) to father;
    father = 0;
  }
  attendre (Token == true);
```

FIGURE 3 – Algorithme Naimi-Tréhel

Question 5

Quels sont les avantages et inconvénients de ce nouvel algorithme par rapport à l'algorithme original de Naimi-Tréhel ?

Question 6

Même question que la précédente par rapport à l'algorithme à jeton de Raymonde vu en cours.