AR (MI048) - Écrit réparti - Première épreuve

L'exercice 1 doit être traité sur une copie séparée.

Exercice(s)

Exercice 1 – Horloges et Causalité

Question 1

On considère l'exécution représentée dans la Figure 1, et les deux relations R_{send} et R_{rcv} définies par :

— $(m_1, m_2) \in R_{send} \quad \Leftrightarrow \quad send(m_1) \rightarrow send(m_2)$

— $(m_1, m_2) \in R_{rcv} \quad \Leftrightarrow \quad receive(m_1) \rightarrow receive(m_2)$

où $\rightarrow$ représente la relation de causalité.

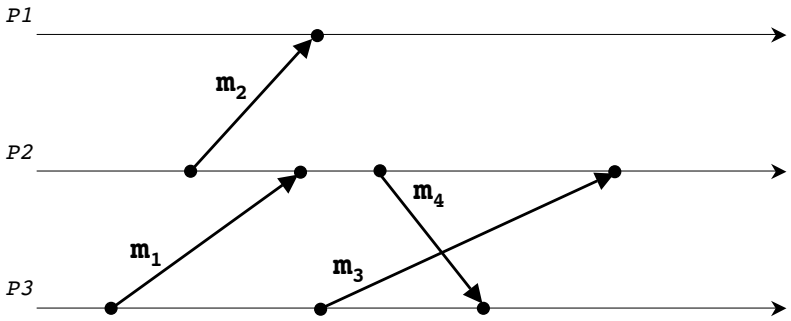

FIGURE 1 – Exécution répartie et causalité

Donnez les valeurs de R_{send} et R_{rcv} pour cette exécution.

Question 2

On rappelle la propriété CO : $\forall m, m'$ *deux messages*, $send(m) \rightarrow send(m') \Rightarrow receive(m) \rightarrow receive(m')$

L'exécution de la Figure 1 vérifie-t-elle la propriété CO ? Justifiez votre réponse.

Peut-on le vérifier au moyen des horloges scalaires (Lamport) ? Au moyen des horloges vectorielles ? Justifiez.

On s'intéresse maintenant à un mécanisme de diffusion qui doit respecter la causalité. Les fonctions **broadcast** et **deliver** sont offertes à l'application. Pour réaliser ce mécanisme, chaque site de l'application gère une file d'attente dans laquelle il stocke les messages lors de leur arrivée. La délivrance d'un message correspond au moment où ce message est extrait de la file d'attente et transmis à l'application. Chaque site de l'application (on considère ici le site p) exécute les deux primitives suivantes :

broadcast(m)

$HV_p[p] = HV_p[p] + 1$;
$\forall r \in Sites \setminus \{p\}$, envoyer($\langle p, m, HV_p \rangle$) à r

recevoir($\langle q, m, HV_m \rangle$)
$\{q \neq p\}$
insérer $\langle q, m, HV_m \rangle$ dans FA_p
attendre jusqu'à ce que :
1. $HV_m[q] = HV_p[q] + 1$
2. $HV_m[k] \leq HV_p[k] \quad \forall k \neq q$

retirer $\langle q, m, HV_m \rangle$ de FA_p
deliver(m)
$HV_p[q] = HV_p[q] + 1$;

FA_p est une file d'attente, initialement vide. HV_p est un vecteur de compteurs initialisé à 0 (vecteur nul). Aucune autre primitive ne manipule le vecteur HV_p.

Après chaque exeécution de **recevoir**, le site p examine la file FA_p pour voir si des messages peuvent être retirés.

Question 3

Quelle interprétation peut-on donner à la valeur HV_p[p] ?
Quelle interprétation peut-on donner à la valeur HV_p[q], $q \neq p$?

Question 4

On considère l'exécution de la Figure 2.

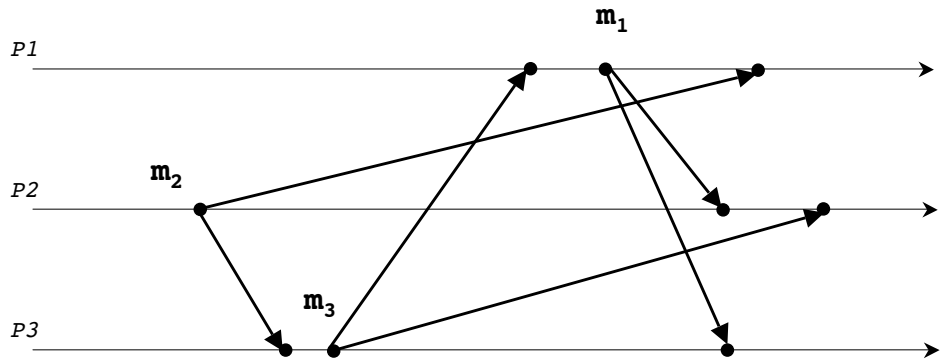

FIGURE 2 – Échange de broadcasts

Pour chaque événement e_i^j (e_i^j est le j$^{\text{ème}}$ événement sur le site i), donnez la valeur du vecteur HV_i et l'état de la file FA_i après l'exécution de l'événement. Lorsque l'événement regroupe plusieurs opérations (plusieurs messages sont extraits de la file), vous préciserez les évolutions liées à chacune de ces opérations.

Question 5

m_3 peut-il être la cause du broadcast de m_1 par l'application ? Justifiez votre réponse.

Exercice 2 – Parcours arbre

Nous considérons une topologie répartie en arbre binaire. Les liens sont bidirectionnels. Chaque processus (site) connaît l'identité de son père et de ses fils. Pour cela, chaque processus possède une variable locale **`int`** `voisin[3]` où :

`voisin[0]` : identité du fils de gauche ou -1 si pas de fils à gauche ;
`voisin[1]` : identité du fils de droite ou -1 si pas de fils à droite ;
`voisin[2]` : identité du père ou -1 si racine ;

Notre but est de offrir un algorithme qui réalise un parcours en profondeur de l'arbre afin qu'à la terminaison de l'algorithme, le site racine affiche les identités de tous les sites de l'arbre en ordre **préfixé** : site, sous arbre gauche, sous arbre droit.

Par exemple, pour l'arbre de la Figure 3.a, le rendu du parcours en ordre préfixé est : 1, 2, 4, 5, 7, 8, 3, 6, 9. Le parcours est montré dans la Figure 3.b.

Le site racine démarre l'algorithme en appelant la fonction *Init_parcours ()* et lorsque un site S_i reçoit un message du site S_j il exécute la fonction *Reception_msg* ($S_j, \langle msg \rangle$)

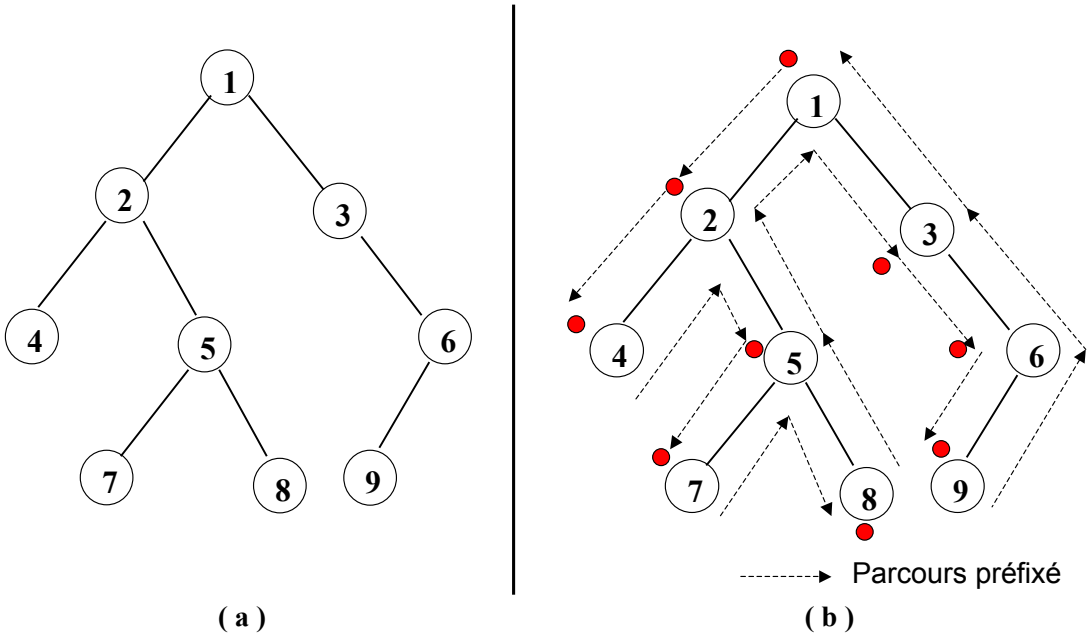

FIGURE 3 – **Parcours en Profondeur préfixé**

Question 1

Donnez le code des fonctions *Init_parcours ()* et *Reception_msg* ($S_j, \langle msg \rangle$) en spécifiant les variables locales que vous utilisez ainsi que la structure du contenu des messages.

Question 2

Votre algorithme peut-il être considéré comme un algorithme total ? Justifiez votre réponse.

Question 3

Quelle est la complexité en nombre de messages de votre algorithme ? Et en terme de temps ? Justifiez vos réponses.

Exercice 3 – Exclusion mutuelle

Le parcours en profondeur de l'arbre binaire en ordre préfixé organise logiquement les sites de l'arbre en anneau comme le montre la Figure 4. Nous voulons donc offrir un algorithme d'exclusion mutuelle basé sur l'algorithme de Le Lann où un jeton tourne en permanence dans l'anneau.

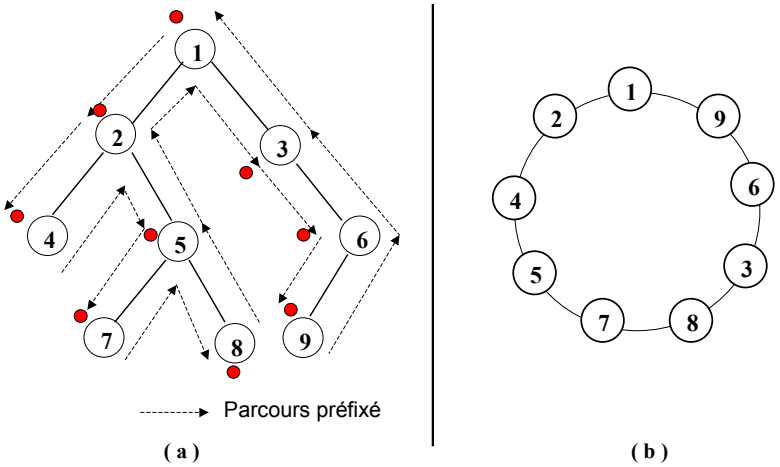

FIGURE 4 – **Organisation de l'arbre en anneau**

Lorsqu'un site reçoit le jeton, s'il veut rentrer en section critique il garde le jeton et rentre en section critique ; sinon il le passe à son successeur dans l'anneau. En sortant de la section critique, il envoie aussi le jeton à son successeur.

`requesting` est une variable booléenne initialisée à `faux` ; `jeton_present` est une variable booléenne initialisée à `faux`, sauf pour le site racine où elle vaut `vrai` ; `pred` et `succ` représentent respectivement le prédécesseur et le successeur du site S_i dans l'anneau logique.

Les fonctions exécutées par le site S_i dans l'algorithme original de Le Lann sont les suivantes :

```
Init () {
  if (voisin[2] != -1)
    jeton_present = false;
  else {
     jeton_present = true;
     if (! requesting ) {
        jeton_present = false;
        Transmettre(<jeton>) to succ;
     }
  }
}

Demande_SC () {
   requesting = true;
   attendre (jeton_present);
}
```

```
Sortie_SC () {
   requesting = false;
   jeton_present = false;
   Transmettre(<jeton>) to succ;
}


Reception_msg(pred,<jeton>) {
      jeton_present = true;

      if (! requesting ) {
          jeton_present = false;
          Transmettre(<jeton>) to succ;
      }
 }
```

Question 1

En recevant le message $\langle jeton \rangle$, le site S_i n'a pas toujours droit d'utiliser le jeton. Par exemple, en recevant le jeton du site 4, le site 2 n'a pas droit de l'utiliser. Modifiez la fonction *Reception_msg* $(pred, \langle jeton \rangle)$ pour que

1. le récepteur du message $\langle jeton \rangle$ ne mette $jeton\_present$ à $true$ que s'il a le droit d'utiliser le jeton ;
2. le récepteur du message transmette le jeton s'il n'a pas le droit de l'utiliser ou s'il n'est pas demandeur de la section critique.

Question 2

Donnez le code de la fonction *Transmettre* ($\langle jeton \rangle$) *to succ.*