UPMC
1881 SORBONNE UNIVERSITÉS

AR (MI048) - Écrit réparti - Première épreuve

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

Exercice(s)

Exercice 1 – Horloges et Causalité

On considère un ensemble de N sites gérant un mécanisme d'horloges. Chaque site i dispose d'un vecteur d'horloges V_i de dimension N, initialisé à 0. La mise à jour des horloges se fait sur le site i selon l'algorithme suivant :

1. à chaque événement sur le site : $V_i[i] = V_i[i] + 1$;
2. lors de l'envoi d'un message : le site i inclut dans le message la valeur de $V_i[i]$ *après* application de la règle (1) ;
3. lors de la réception d'un message en provenance du site j contenant la valeur h : $V_i[j] = h$ et application de la règle (1).

Les communications sont fiables et vérifient la propriété FIFO.

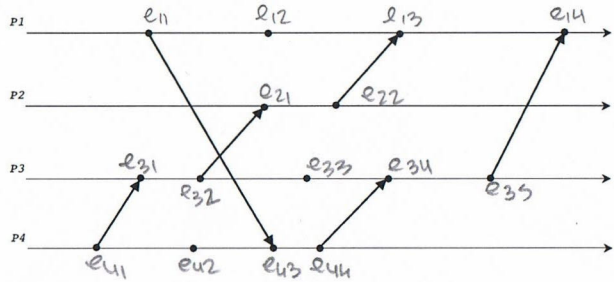

FIGURE 1 – Horloges et causalité

Question 1

Donnez l'exécution de cet algorithme sur le schéma de communications de la Figure 1.

Question 2

Ce mécanisme d'horloges est-il consistant avec (capture-t-il) la causalité ? Fortement consistant (la caractérise-t-il) ? Justifiez précisément vos réponses.

Question 3

Soit V_i^h le vecteur d'horloge du site i au moment où $V_i[i] = h$. On considère l'algorithme suivant, appliqué sur une trace d'exécution comme celle de la Figure 1. Cet algorithme calcule, pour un événement d'horloge h sur le site i, le vecteur CB associé.

```
Init(i : process ; h : horloge) {
    Pour k de 1 à N {
        CB[k] = 0 ;
    }
    CB[i] = h ;
    Visit_Event(i, h) ;
}
```

```
Visit_Event(i : process ; h : horloge) {
    Pour k de 1 à N, k ≠ i {
        a = V_i^h[k] ;
        Si a > CB[k] {
            CB[k] = a ;
            Visit_Event(k, a) ;
        }
    }
}
```

Donnez le déroulement de l'appel `Init(1,4)` sur la trace d'exécution que vous avez calculée à la question 1, en précisant les modifications du vecteur CB à chaque étape ainsi que les appels récursifs effectués.

Question 4

Quel commentaire peut-on faire sur la valeur finale du vecteur CB associé à un événement ?

Question 5

L'exécution de la Figure 1 vérifie-t-elle la propriété CO ($\forall m, m',\ \mathit{send}(m) \rightarrow \mathit{send}(m') \Rightarrow \mathit{rcv}(m) \rightarrow \mathit{rcv}(m')$) ? Justifiez précisément votre réponse.

Exercice 2 – Exclusion mutuelle

Nous voulons traiter le problème de la k-exclusion mutuelle en utilisant k jetons. Dans un premier temps, nous considérons une approche **centralisée** où un processus particulier appelé maître gère les requêtes d'accès. Ce processus ne fait pas de requêtes. Pour chaque jeton, il y a une file de requête pendante **répartie** associée (comme la file de next de l'algorithme de Naimi-Tréhel). Notez que le maître ne garde pas localement les files mais seulement l'information sur le dernier demandeur de chaque file. A chaque requête reçue, le maître doit décider quel jeton, parmi les k, sera affecté au processus demandeur. Ce choix est fait de façon circulaire (round-robin), au moyen de la variable `next` qui détermine le numéro du prochain jeton à utiliser.

Afin de gérer les informations décrites ci-dessus, le maître possède la variable locale suivante :

```
struct info_file {
   int last[k] ;  /* sauvegarde le dernier demandeur pour chaque jeton */
   int  next ;  /* next file à choisir; mise à jour  File.next=(File.next+1) % k ; */
 } File ;
```

Question 1

Proposez un algorithme pour gérer cette k-exclusion mutuelle. Donnez le code des fonctions *DemandeSC ( )*, *SortieSC()*, *ReceptionMSG(msg,S_j)* (pour le maître et processus demandeurs où S_j est le site émetteur), ainsi que les types de messages. Spécifiez les messages et leur contenu. Vous pouvez ajouter d'autres variables locales au maître et/ou processus demandeurs, si nécessaire. Donnez aussi les initialisations des variables. Notez que maître ne traite que les messages de requêtes.

Observation : Un processus ne doit jamais envoyer un message à lui-même.

Nous voulons maintenant modifier l'algorithme de la question précédente pour avoir une approche **répartie**. Dans ce cadre, le maître n'est plus un processus particulier mais le rôle du maître tourne entre les processus : il sera toujours affecté au dernier processus demandeur. Autrement dit, le *maître en cours* considérera comme prochain maître le premier processus dont il a reçu une requête. Observez que le processus maître peut lui aussi émettre des requêtes. Pour que la requête d'un processus puisse atteindre le maître en cours, nous utiliserons l'approche de l'arbre dynamique (l'arbre de *father*) proposée par Naimi-Tréhel. Par conséquent, le maître en cours est la racine de l'arbre de *father* et doit gérer les informations de la File de type `struct info_file` (que le maître précédent lui enverra).

Question 2

Il est possible que le prochain maître reçoive des requêtes d'entrée en section critique avant qu'il ait les informations de la File `struct info_file`. Donnez un exemple qui illustre ce scénario.

Question 3

La fonction *SortieSC()* ne change pas par rapport à l'algorithme centralisé. Spécifiez les messages et leur contenu utilisés par cette nouvelle version. Vous pouvez ajouter d'autres variables locales, si nécessaire. Donnez les initialisations de ces variables. Vous devez aussi soit donner le code des fonctions *DemandeSC ( )* et *ReceptionMSG(msg, S_j)* soit les décrire en détail.

Question 4

Quels sont les avantages et/ou inconvénients de cette approche par rapport à celle de Raymonde étudiée en TD qui adapte l'algorithme de Ricart-Agrawala à la k-exclusion mutuelle ?

