# AR (MI048) - Écrit réparti - Première épreuve

*Les deux exercices doivent être traités sur des copies séparées.*

**Exercice(s)**

## Exercice 1 – Horloges et Causalité

On s'intéresse à une application répartie composée de 3 processus échangeant des messages. Une exécution de cette application est représentée dans la figure 1.

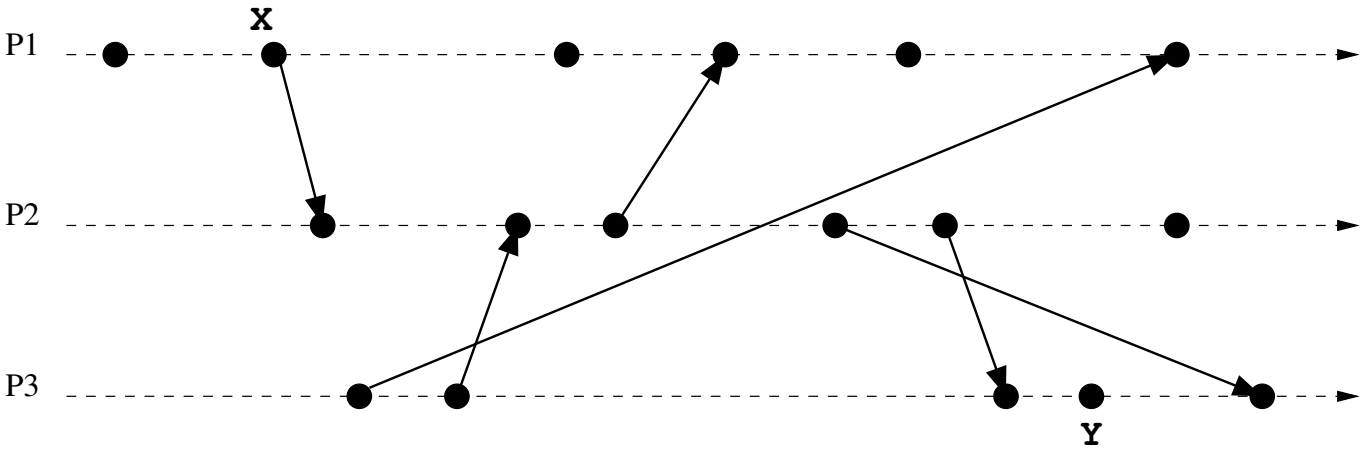


FIGURE 1 – Exécution d'une application distribuée

**Question 1**

Donnez les horloges scalaires affectées aux événements, ainsi que les valeurs transportées par les messages lors de cette exécution.

**Question 2**

Donnez les horloges vectorielles affectées aux événements avec l'algorithme de Mattern, ainsi que les valeurs transportées par les messages lors de cette exécution.

**Question 3**

Donnez les horloges vectorielles affectées aux événements avec l'algorithme de Singhal et Kshemkalyani, ainsi que les valeurs transportées par les messages lors de cette exécution.

**Question 4**

A-t-on $X \rightarrow Y$ (cf. figure 1) ? Justifiez votre réponse, si nécessaire en vous aidant du dessin de l'exécution.

### Question 5

Peut-on déduire votre réponse à la question précédente des valeurs d'horloges calculées avec l'algorithme de Mattern (cf. votre réponse à la seconde question) ? Même question avec l'algorithme de Singhal et Kshemkalyani (cf. votre réponse à la troisième question) ? Justifiez vos deux réponses.

### Question 6

A partir des valeurs obtenues à la question 2 (algorithme de Mattern), complétez l'exécution avec les événements dont les valeurs d'horloges sont les suivantes (ne pas reporter les valeurs d'horloges calculées à la question 2 sur votre réponse) :

$$\begin{bmatrix} 2 \\ 7 \\ 4 \end{bmatrix} \quad \begin{bmatrix} 5 \\ 8 \\ 4 \end{bmatrix} \quad \begin{bmatrix} 7 \\ 6 \\ 2 \end{bmatrix}$$

S'il y a lieu, ajoutez les messages à votre réponse.

## Exercice 2 – Exclusion Mutuelle

Nous considérons un système distribué composé d'un ensemble de $N$ processus fiables $\prod = \{p_1, p_2, ...p_N\}$. Il y a un seul processus par site ou nœud.

Les processus ne partagent pas de mémoire et communiquent uniquement par passage de messages. Les nœuds sont supposés être connectés par des liens de communications point à point fiables (ni perte ni duplication) et FIFO. Le graphe de communication est considéré complet, *i.e.*, tout processus peut communiquer avec n'importe quel autre processus.

Le système contient un ensemble $R = \{r_1, r_2..., r_m\}$ de $m$ ressources différentes. Aucun ordre n'est défini sur $R$.

Les algorithmes d'exclusion mutuelle généralisée permettent de gérer les accès concurrents des processus sur un ensemble de ressources partagées. Autrement dit, un processus peut demander un sous-ensemble de $k$ ressources ($k \leq m$). Par exemple, le processus $p_1$ peut demander les ressources $r_1$ et $r_3$ tandis que $p_2$ les ressources $r_2$, $r_3$ et $r_5$. Pour demander l'accès à $k$ ressources, l'application appelle la fonction $Request\_CS(D_k)$ où $D_k$ est un ensemble de $k$ ressources et pour les libérer la fonction $Release\_CS()$. La section critique concerne, par conséquent, l'accès à toutes les ressources du sous-ensemble demandé. De plus, un processus peut initier une nouvelle requête si et seulement si la requête qu'il a précédemment demandée a été satisfaite, c'est-à-dire que la section critique s'est terminée par l'exécution de $Release\_CS()$. Par conséquent, il y a au plus $N$ requêtes pendantes dans le système.

### Question 1

Supposons que chaque ressource $r_i$ est gérée par une instance d'un algorithme d'exclusion mutuelle classique (exemple : Ricart-Agrawala, Suzuki-Kasami, Naimi-Tréhel, etc.). Une instance qui gère la ressource $r_i$ offre donc les fonctions $Request\_CS_i()$ et $Release\_CS_i()$. La fonction $Request\_CS(D_k)$ et $Release\_CS()$ consisteraient alors à l'appel à un ensemble de fonctions $Request\_CS_i()$ et $Release\_CS_i()$ respectivement pour $i$ quelconque $\in D_k$. Quel problème peut-il arriver avec une telle solution ?

On considère les mêmes hypothèses qu'à la question précédente, mais on définit maintenant un ordre total $\prec$ sur l'ensemble des ressources $R$ où $r_i \prec r_j$ si $i < j$. Désormais, les processus doivent demander les ressources requises de $D_k$ en respectant l'ordre $\prec$.

### Question 2

Ceci résout-il le problème de la question 1 ? Pourquoi ?

Nous reprenons à partir de maintenant **les hypothèses initiales décrites au début de l'exercice** et que par conséquent, il n'y a plus d'ordre total défini sur $R$.

Nous proposons un algorithme qui se caractérise par les propriétés suivantes :

- chaque ressource $r_i$ est :
  - représentée par un unique jeton $t_i$
  - associée à une file d'attente distribuée dont le premier élément est le possesseur du jeton
- Avant de demander un quelconque jeton de ressource $t_i$, un processus doit demander l'acquisition d'un **jeton unique de contrôle** noté $T_c$. Ce jeton peut être vu comme une ressource partagée auxiliaire et son accès exclusif est géré avec l'algorithme de Naimi-Tréhél.
- Le jeton de contrôle contient un vecteur $VT$ de $m$ entrées ($m$ est égal au nombre de ressources du système). À chaque ressource $r_i$, l'entrée correspondante indique :
  - soit l'identifiant du dernier site demandeur de $r_i$
  - soit la valeur $nil$ qui indique que le jeton $t_i$ est libre et est inclus dans $T_c$ (aucun processus ne possède ou n'a demandé $t_i$)
- À la réception du jeton de contrôle, un processus prend toute ressource requise incluse dans $T_c$ et envoie pour chaque jeton manquant, un message INQUIRE au dernier demandeur. Chaque entrée correspondante à une ressource requise dans $VT$ est alors mise à jour en y affectant l'identifiant du processus courant qui est devenu le

dernier demandeur. Le jeton de contrôle peut désormais être libéré et renvoyé à un prochain demandeur éventuel. Toutefois avant la libération, le processus met dans $VT$ toutes les ressources non utilisées qu'il possède.

- À la réception d'un message INQUIRE pour un jeton $t_i$ :
  - si le site receveur possède $t_i$ et ne l'utilise pas, il envoie immédiatement $t_i$ à l'émetteur par un message ACK
  - sinon si le jeton est utilisé ou pas encore possédé, l'envoi de ACK est ajourné jusqu'à ce que le receveur libère $t_i$ lors de l'appel à $Release\_CS()$.
- Initialement toute ressource est incluse dans le jeton de contrôle. Ce dernier est détenu par le processus $p_1$ qui est par conséquent la racine ($father = nil$) de l'arbre dynamique gérant $T_c$. La variable $father$ de tous les autres processus est donc égale à $p_1$.

Les types de messages de cet algorithme sont donc :
- $REQ\_CONTROL$ : un message pour demander le jeton de contrôle
- $TOKEN\_CONTROL$ : un message transmettant le jeton de contrôle
- $INQUIRE$ : un message demandant un jeton ressource
- $ACK$ : un message transmettant un jeton de ressource

## Question 3
Quel est le rôle du jeton de contrôle dans l'algorithme ?

## Question 4
Supposons un système réparti avec 4 processus ($p_1$, ..., $p_4$) et 5 ressources ($r_1$, ..., $r_5$). Nous considérons que juste après l'initialisation du système, les processus exécutent les primitives $Request\_CS$ dans l'ordre suivant :

1. $p_2$ : $Request\_CS(\{r_1, r_3\})$ ;
2. $p_3$ : $Request\_CS(\{r_4\})$ ;
3. $p_4$ : $Request\_CS(\{r_1, r_2, r_4\})$ ;

**Donnez à l'état initial et après chaque exécution des primitives** l'état du système à savoir :
- l'arbre dynamique gérant $T_c$
- le contenu du vecteur du jeton de contrôle
- le possesseur de $T_c$
- les possesseurs des jetons de ressource ainsi que les files distribuées associées.

De plus, **donnez les différents transferts de messages** (type, expéditeur, destinataire) entre chaque état. En tout 4 états vous sont donc demandés. Il vous est recommandé de faire un schéma pour chaque état.

## Question 5
Quel est le nombre de messages échangés de l'algorithme lorsqu'un processus demande $k$ ressources ? Pour votre calcul, prenez en compte l'obtention du jeton de contrôle, plus l'accès au ressources.

## Question 6
Expliquez informellement comment la propriété de sûreté de l'algorithme est assurée.

## Question 7
Nous souhaitons maintenant avoir le pseudo-code de cet algorithme. Pour cela nous vous demandons de :
- Donner les variables locales à chaque processus ainsi que la procédure d'initialisation.
- Spécifier les contenus de chaque type de message.
- Donner le pseudo-code des fonctions $Request\_CS(D_k)$ et $Release\_CS()$.
- Donner pour chaque type de message la primitive gérant la réception du message en question.