DEA d'Informatique - Systèmes et Communication. Année 2001-2002
INPG-UJF Grenoble

# Algorithmique et Techniques de Base des Systèmes Répartis
# Examen

20 décembre 2001
Durée: 2 heures, documents autorisés

---

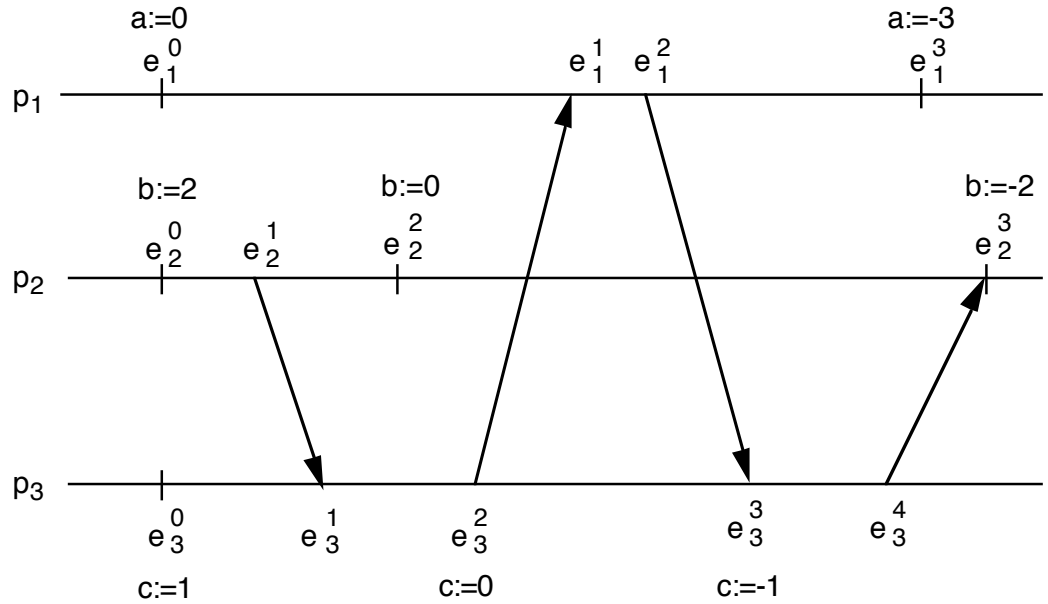
*L'examen comporte 3 parties indépendantes. Lire l'ensemble des énoncés avant de commencer à répondre. La longueur des énoncés n'est pas signe de difficulté mais elle est nécessaire à une bonne spécification des problèmes. La **clarté**, la **précision** et la **concision** des réponses, ainsi que leur **présentation matérielle**, seront des éléments importants d'appréciation.*

---

**Prière de rédiger les réponses des 3 parties sur 3 feuilles séparées.**

## Partie 1 ( 7 points)

On considère un calcul réparti comportant 3 processus p1, p2, p3, représentés par la figure ci-après.




**Question a)** Pour chacun des événements indiqués, donner la date de cet événement, d'abord dans un système d'horloges scalaires, puis dans un système d'horloges vectorielles. L'état initial du système est ($e_1^0$, $e_2^0$, $e_3^0$). La valeur initiale de toutes les horloges, dans cet état initial, est fixée par convention à 0 (resp. (0,0,0))

**Question b)** Construire et représenter graphiquement le treillis des états cohérents correspondant au calcul réparti ci-dessus.

**Question c)** Chacun des processus $p_1$, $p_2$, $p_3$, met respectivement à jour une variable $a$, $b$, $c$, comme indiqué. Entre les mises à jour successives, chaque variable conserve sa dernière valeur. Trouver un prédicat fonction des trois variables $a$, $b$, $c$, qui soit *stable* pour le calcul considéré, et qui ne soit pas vérifié dans l'état initial. On rappelle qu'un prédicat $P$ est stable s'il vérifie la propriété suivante : si $P$ est vrai dans un état cohérent $S$, il est vrai dans tout état cohérent atteignable depuis $S$.

**Question d)** On s'intéresse maintenant aux prédicats *non stables*. Pour un tel prédicat $P$, on définit deux propriétés.

*Pos (P)* : il existe *une* observation cohérente du calcul réparti (i. e. un chemin dans le treillis des états cohérents) telle que $P$ soit vérifié dans un état de cette observation (un point dans le treillis).

*Def (P)* : Pour *toute* observation cohérente du calcul réparti, il existe un état dans lequel $P$ est vérifié.

Pour le calcul considéré, on demande de trouver (s'il existe) un prédicat $Q$ fonction des variables $a$, $b$, $c$, non stable et vérifiant *Pot (Q)*, et un prédicat $R$ fonction des variables $a$, $b$, $c$, non stable et vérifiant *Def (R)*.

---

# Partie 2 (7 points)

### 2.1 Question de cours

Rappeler les deux principales propriétés attendues d'un détecteur de défaillances. Quelles sont les quatre classes principales de détecteurs de défaillance complets que l'on peut identifier ? Quelle est la classe la plus faible permettant de résoudre le problème du consensus simple en présence de pannes franches ? Sous quelle hypothèse concernant le nombre de fautes ?

**Note** : dans tous les exercices qui suivent, on veillera dans l'écriture d'un algorithme à détailler les variables d'état maintenues par l'algorithme, leur initialisation, et les actions effectués par chaque processus. On pourra utiliser les primitives du pseudo-code présenté en cours.

### 2.2 Exercice 1

On considère un réseau fiable (pas de perte ni de corruption ou de duplication de messages) avec délais de transmission bornés par une constante D connue, et des processus susceptibles de pannes franches uniquement. Donner un algorithme simple implantant un détecteur de défaillance parfait, i.e. complet et fortement précis.

Justifier simplement que l'algorithme réalise un détecteur parfait.

### 2.3 Exercice 2

On considère un réseau fiable, avec délais de transmission bornés par une constante D inconnue, et des processus susceptibles de pannes franches uniquement. Donner un algorithme simple implantant un détecteur de défaillance finalement parfait, i.e. complet et finalement fortement précis. Justifier simplement que l'algorithme réalise un tel détecteur.

Dans les mêmes conditions, est-il possible d'implanter un détecteur parfait ? Justifier votre réponse (positive ou négative)

### 2.4 Exercice 3

L'algorithme de diffusion causale de Hadzilacos et Toueg présenté en cours s'appuie sur une diffusion fiable FIFO de listes de messages. Dans les mêmes conditions d'environnement (réseau fiable, pannes franches), donner un algorithme de diffusion causale utilisant uniquement une primitive de diffusion causale fiable et qui évite la diffusion de listes de messages.

[Suggestion : on pourra considérer l'utilisation d'estampilles associées aux messages]

### 2.5 Exercice 4

On considère le problème de l'attaque coordonnée (protocole de coordination entre deux processsus qui doivent parvenir à un accord en échangeant des messages). On suppose que les processus n'ont pas de défaillance.

a) On suppose que le système de communication peut perdre des messages. Rappeler le raisonnement qui montre l'impossibilité de construire un protocole de coordination en un nombre fini d'étapes.

b) On suppose maintenant que le système de communication a les propriétés suivantes : un message peut se perdre, mais l'émetteur en est averti et peut le réémettre ; au bout d'un nombre fini (mais non borné) de réémissions, le message finit par parvenir à destination. Le raisonnement du a) est-il encore valide ? Explicitez votre réponse.

## Partie 3 (6 points)

Vous devez réaliser un système à nommage relatif comme vu en cours (la répartition n'est pas prise en compte dans ce problème). Vous disposez des notations et primitives suivantes.

Un objet en mémoire d'exécution est désigné par une référence de type ***Object***.

On gère des descripteurs d'objet de classe ***Desc***. Un descripteur contient une localisation globalement unique. Il contient deux champs ***partition*** et ***localName***, donnant globalement la partition de localisation de l'objet et son nom local à cette partition. On stocke un descripteur en tête de chaque objet, ce descripteur contenant la localisation de l'objet.

Un lien de poursuite est un objet ne contenant que ce descripteur.

On peut utiliser les fonctions suivantes sur un objet chargé en mémoire d'exécution :

***Desc D (Object)*** : cette fonction retourne le descripteur d'un objet donné (en mémoire).

***Boolean FW (Object)*** : cette fonction indique si un objet donné est un lien de poursuite.

On dispose de fonctions permettant de manipuler les partitions en mémoire de stockage.

***Object load (partition, localName)*** : cette fonction charge en mémoire d'exécution un objet situé dans une partition donnée, à partir de son nom local à cette partition.

***void store (partition, localName, obj)*** cette fonction met à jour l'objet de nom local ***localName*** dans la partition ***partition***, avec la valeur de l'objet référencé par ***obj***.

***localName create (partition, obj)*** cette fonction crée un nouvel objet dans une partition et retourne le nom local alloué.

Pour l'écriture des algorithmes, utiliser un langage "pseudo-C" ou "pseudo-Java", i.e. en prenant les notations utiles mais sans nécessairement utiliser une syntaxe stricte.

**Question 1.** Écrire l'algorithme de la fonction ***Object loadObject (name)*** qui charge un objet à partir d'un nom ***name*** contenu dans une variable de l'objet courant en mémoire d'exécution. On suppose que l'objet courant, dans lequel le programme s'exécute, est désigné par la variable ***this***.

**Question 2.** Écrire l'algorithme de la fonction ***void migrate (obj, partition)*** qui déplace l'objet ***obj*** (***obj*** est sa désignation de type ***Object*** en mémoire) vers la partition ***partition***.

**Question 3.** Modifier l'algorithme de la question 1 pour que la localisation de l'objet à charger raccourcisse les liens de poursuite créés par les migrations.