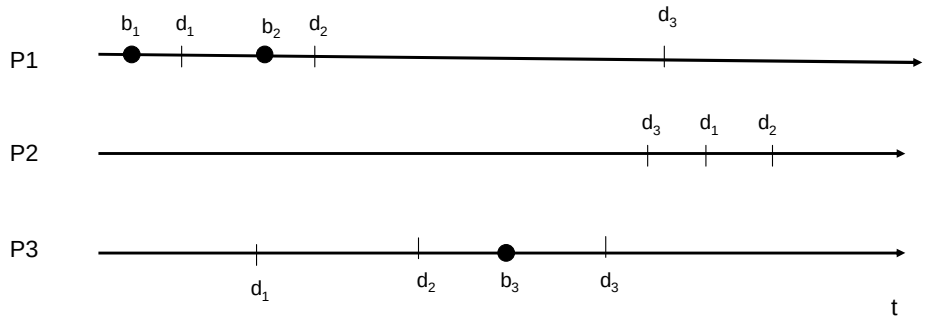
ARA – Nov. 2009
Durée : 2 heures
Documentation autorisée : polycopie du cours
L.Arantes, O.Marin, P.Sens

DIFFUSION FIABLE, MODELE DE COHERENCE ET REGISTRE

Exercice 1: Diffusion fiable

Considérez le scénario ci-dessous :



● b_i = diffusion (broadcast) of message i

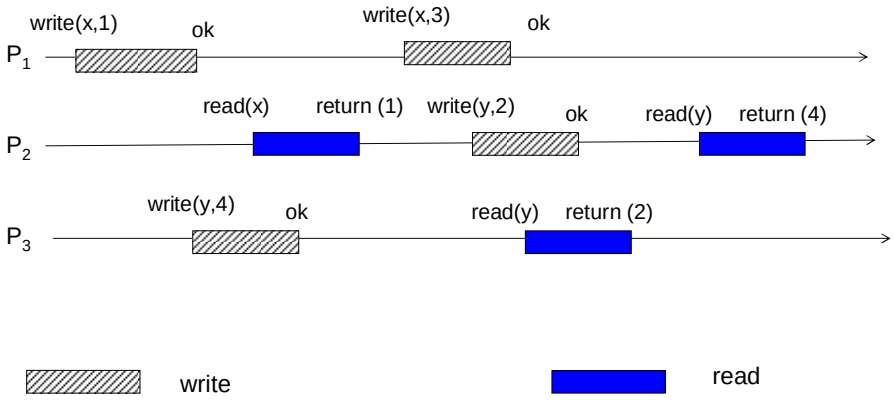
| d_i = délivrance correspondante à la diffusion du message i

1.1

Est-ce que le scénario représente une diffusion FIFO, causale ou totalement ordonnée ? Justifiez votre réponse par rapport à chacun de ces types de diffusion.

Exercice 2: Modèle de Cohérence

Considérez le schéma ci-dessous :



2.1

Est-ce qu'il s'agit d'une cohérence séquentielle? D'une cohérence causale ? D'une cohérence PRAM ? Justifiez votre réponse pour chacun de ces trois types de cohérence.

Exercice 3: REGISTRE

Considérez un registre ***binaire*** R' du type «safe» et MRSW (multiples lecteurs, un écrivain). Les seules valeurs valables pour un registre binaire sont 0 et 1. Le processus (thread) P_0 (écrivain) peut écrire sur R' et P_1,..P_n (lecteurs) peuvent lire son contenu. Si une lecture est concurrente avec une écriture la lecture peut rendre 0 ou 1.

Nous voulons construire un registre ***binaire*** R du type «regular» et MRSW (multiples lecteurs, un écrivain) en utilisant R'. Si une lecture sur R est concurrente avec une écriture, la lecture rend soit la valeur en train d'être écrite soit la dernière valeur écrite.

3.1

Donnez le code des deux fonctions de R:

- *boo Write(R,val) :* écrit la valeur val sur R et renvoie OK (1) lors de la fin de l'écriture.
- *int Read (R) :* renvoie valeur de R.

Indiquez les variables locales utilisées, si nécessaire.

3.2

Est-ce que votre solution pourrait être appliquée dans le cas où les registres R' et R ne sont pas du type binaire mais du type entier ? Justifiez votre réponse.

Exercice 4: Détecteur de fautes

On considère, un ensemble de processus $\Pi = \{p_1, p_2, .., p_n\}$ communiquant par messages. Les liens de communication forment un graphe complet (chaque processus peut directement envoyer un message à chacun des processus). Les liens de communication sont fiables.

On considère des fautes de type « crash ». Le réseau est considéré partiellement synchrone : après le temps GST (inconnu), il est existe des bornes sur les délais de transmissions et de traitement des messages et tous les messages émis avant GST ont été reçus.

Soit le détecteur de fautes de classe Ω associé à p_i dont le code est le suivant :

1. **Initialisation**
2. $leader_i = 1$
3. Pour tout j : $\Delta_{i,j}$ = Timeout par défaut

4. **Tâche 1 :**
5. Boucle infinie
6. Si $leader_i = i$
7. Pour *k* = i+1 à *n*
8. Envoyer I-AM-ALIVE à p_k
9. Attendre Δ_H
10. Fin boucle

11. **Tâche 2 :**
12. Boucle infinie
13. Si $leader_i < i$
14. Attendre $\Delta_{i,leader_i}$
15. Si $received_i$ alors
16. $received_i$= false
17. sinon
18. $leader_i = leader_i+1$
19. Fin boucle

20. **Tâche 3 :**
21. Boucle infinie
22. Réception I-AM-ALIVE de p_j
23. Si $j = leader_i$
24. $received_i$= true
25. Sinon si $j < leader_i$ alors
26. $leader_i = j$
27. $received_i$ = true
28. Fin si
29. Fin boucle

4.1

En absence de faute et si tous les messages arrivent avant l'expiration des temporisateurs, quel est le nombre total de messages échangé à chaque itération (rondes).

4.2

Dans quelle(s) configuration(s) un processus k (k>1) peut se proposer comme leader.

4.3

Dans quels cas un processus p_x peut redevenir leader alors que le leader courant est un processus p_y (y>x) ?

4.4

Pourquoi malgré l'hypothèse partiellement synchrone, cet algorithme peut ne pas implémenter un Ω?

4.5

Modifier l'algorithme pour qu'il puisse implémenter un Ω. Indiquez seulement les lignes à ajouter et/ou modifier.

4.6

On considère p_k, le processus correct de plus petit identifiant. Montrez qu'il existe un temps t_0 à partir duquel $leader_k = k$.

4.7

Donnez une preuve informelle (en quelques lignes) montrant qu'il existe un temps à partir duquel pour tout i $leader_i = k$, p_k étant le processus correct de plus petit identifiant.

TECHNIQUES DE REPRISE SUR SAUVEGARDE

Exercice 5: Points de reprise non coordonnés

Les figures 1 et 2 représentent des exécutions d'applications, au cours desquelles des sauvegardes locales ont été effectuées sur disque de manière indépendante et spontanée par chacun des processus A, B, C, et D. Les points représentent des sauvegardes locales et les flèches des messages échangés au cours de chaque exécution.

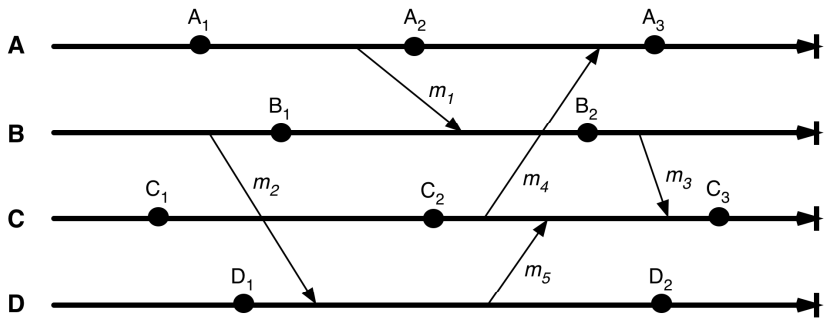


Fig. 1 — Exécution de l'application 1

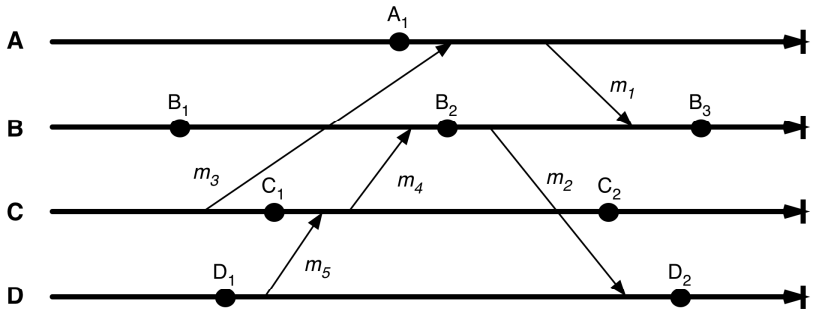


Fig. 2 — Exécution de l'application 2

5.1

Donnez, pour chacune des exécutions, tous les Z-paths présents qui contiennent au moins deux messages. Existe-t-il des points de sauvegarde qui ne pourront faire partie d'aucune ligne de recouvrement ? Si oui, lesquels ?

Une panne de courant interrompt soudainement les deux applications sur l'ensemble des nœuds, à l'endroit où les lignes temporelles associées à l'exécution des processus sont sectionnées. Au moment de redémarrer, on cherche à établir la coupure cohérente la plus avancée dans le temps.

5.2

Identifiez les ensembles de points de sauvegarde qui correspondent à la coupure cohérente la plus avancée dans le temps pour chacune des exécutions représentées sur les figures 1 et 2.

Exercice 6: Points de reprise coordonnés

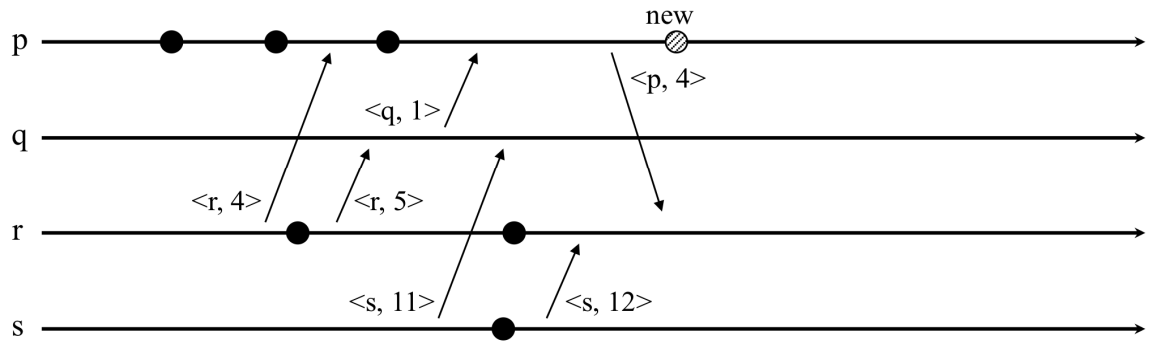
On s'intéresse à l'algorithme de Koo & Toueg 87, dont voici une implémentation possible :

```
int smsg = 0; // message emission counter
int last_rmsg[k] = 0 for every node k; // last message received from k since
last local checkpoint
int first_smsg[k] = 0 for every node k; // first message sent to k since last
local checkpoint
int last_smsg[k] = 0 for every node k; // last message sent to k


{send message to node j}
  smsg++;
  if (first_smsg[j] == 0)
         first_smsg[j] = smsg;
  last_smsg[j] = smsg;
  send(<MSG, i, smsg>, j);


{receive message <MSG, j, msg_nb> from node j}
  last_rmsg[j] = msg_nb;


{initiate backup}
```

```
for every node k {
       if (last_rmsg[k] != 0)
              send(<BACKUP, i, last_rmsg[k]>, k);
       last_rmsg[k] = 0;
       first_smsg[k] = 0;
}
save_state();


{receive message <BACKUP, j, msg_nb> from node j}
  if ((first_smsg[j] != 0) && (first_smsg[j] <= msg_nb))
       initiate_backup();
```

6.1

Complétez le diagramme ci-dessous, avec le processus P comme initiateur :



6.2

Donnez le pseudo-code exécuté par chaque nœud lors d'un recouvrement. Vous pourrez utiliser la primitive `load_state()` qui permet de redémarrer un processus depuis son dernier point de reprise.

6.3

Koo & Toueg 87 prend pour hypothèse que les canaux sont FIFO. Fournissez un exemple simple d'exécution sur des canaux non FIFO, dans lequel la ligne de recouvrement obtenue est incohérente.

6.4

Sans aller jusqu'à l'implémenter, proposez une extension de Koo & Toueg 87 qui peut fonctionner avec des canaux non FIFO. Prenez soin toutefois de détailler les variables à ajouter localement ainsi que leur utilisation.