TD Mémoire Partagée
ARA
Octobre 2020

Exercice 1 – Exclusion Mutuelle

Considérez l'algorithme de la boulangerie de Lamport vu en cours :

Shared Variables:

```
boo choosing[n];
int timestamp[n];
```

Initialization:

```
choosing[1..n] := 0;
timestamp[1..n] := 0;
```

Entry CS Code:

```
1:choosing [i] := 1;
2: timestamp[i] := 1 + max1..n( );
3: chosing[i] := 0;

4: for j := 1 to n do {
5:   await(choosing[j]==0 );
6:   await (timestamp[j] == 0 or (timestam[i],i < timestamp[j],j)) ;
```

Exit Code:

```
7: timestamp[i] := 0;
```

L'algorithme comporte 4 phases:

- PHASE1 : Attribution d'un numéro d'ordre (ticket).
- PHASE2 : Attente de son tour avant l'entrée en section critique.
- PHASE3 : Section critique
- PHASE4 : Sortie de la section critique.

1.1

En sortant de la section critique, processus *i* met timestamp [i] à 0 (ligne 7). Est-ce que la valeur de *timestamp[i]* peut être bornée (*n* processus au maximum) ou doit-elle être non-bornée? Justifiez votre réponse.

Considérez les phrases suivantes :

1. Si les processus *i* et *k* se trouvent dans la PHASE2 de l'algorithme de la boulangerie et *i* est rentré dans la PHASE2 avant que *k* soit rentré dans la PHASE1, alors *timestamp[i] < timestamp[k]*.

2. Si le processus *i* se trouve en section critique (PHASE3) et *k* se trouve dans la PHASE2 de l'algorithme, *alors timestamp[i] < timestamp[k]*.

1.2

Est-ce que les phrases sont-elles vraies ? Justifiez votre réponse.

On considère l'algorithme suivant pour la fonction *max* appelée à la ligne 2 de l'algorithme de la boulangerie :

```
1:  int k=i ;
2: for j=1 to n do
3: if (timestamp[k] < timestamp[j])
4:       k=j;
5: return (timestamp[k]);
```

1.3

Deux processus *i* et *j* peuvent avoir le timestamp: *timestamp[i]=timestamp[j]*. En se basant sur le code de la fonction *max* ci-dessus, expliquez comment cette égalité peut arriver. Est-t-elle un problème? Justifiez votre réponse.

1.4

En donnant un scénario d'exécution, expliquez pourquoi l'implémentation ci-dessus de la fonction *ma*x n'assure pas la sûreté de l'algorithme de la boulangerie ?

1.5

Proposez un algorithme pour la fonction *max* qui enlève le problème de l'algorithme précédent.

Nous voulons maintenant implémenter un algorithme de exclusion mutuelle, basé sur des timestamp (ticket) comme celui de la boulangerie, mais en utilisant qu'un registre du type *read-modify-write* et des variables locales. On considère N processus.

```
function read-modify-write (r :register, f :function)
{
  temp =r;
  r=f(r);
  return (temp);
}
```

1.6

Donnez le code d'un tel algorithme.

Exercice 2: Modèle de Cohérence

Considérez l'exécution de la Figure 1 :

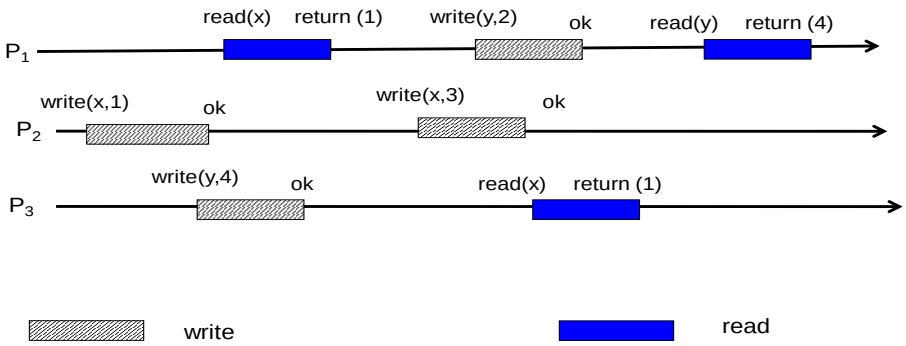

Figure 1

2.1

A quel type de cohérence ce schéma correspond-t-il ? Justifiez votre réponse.

Considérez l'exécution de la Figure2 :

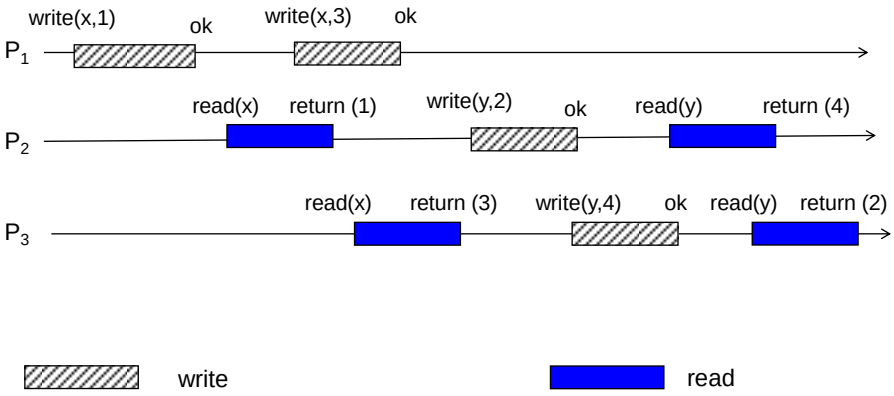

Figure 2

2.2

Quelles sont les opérations d'écritures liées causalement ?

2.3

Encore par rapport à la Figure 2. Est-ce qu'il s'agit d'une cohérence séquentielle? D'une cohérence causale ? D'une cohérence PRAM ? Justifiez votre réponse pour chaqu'une de ces trois types de cohérence.

L'algorithme, vu en cours (transparent 14), qui émule une variable partagée ***x*** linéarizable dans un système par passage de message nécessite d'une diffusion totalement ordonnée tant pour les opérations d'écriture que pour les opérations de lecture.
On considère plusieurs processus écrivains et lecteurs.

Le code de l'algorithme est donné ci-dessous :

2.4

Modifiez l'algorithme pour que l'émulation de la variable partagée *x* respecte la cohérence séquentielle au lieu de la « linéarizabilité ». Justifiez vos choix.

```
int x;
upon operation(op,val) from application
   /* Write */
if (op == write)
   total_order_broadcast (op, id_i);
else
   return x;
```

```
upon reception of message <write, val, id>
   x=val;
   if (id = id_i)
      /* own request which was broadcast */
      return ack to application
```

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.