# Conception des systèmes concurrents

1h45, documents autorisés

2IR, novembre 2016

Les exercices sont indépendants. Le barème est indicatif. Le total des points sera écrêté à 20 si nécessaire.
*Conseil* : il est préférable de **lire attentivement et intégralement** l'énoncé chaque question avant de commencer à y répondre.

## 1 Exécution concurrente (5 points)

On considère une fonction $f$ définie de $\mathbb{Z}$ dans $\mathbb{Z}$ ($\mathbb{Z}$ étant l'ensemble des entiers relatifs) pour laquelle il existe un unique entier $i_0$ tel que $f(i_0) = 0$.

Les 5 algorithmes ci-dessous utilisent deux processus concurrents $P$ et $Q$ pour rechercher $i_0$. Pour que l'un de ces algorithmes soit considéré comme correct, il faut qu'il garantisse que, dans tous les cas, $i_0$ sera trouvé, et que ses **deux** processus seront **terminés** une fois que $i_0$ aura été trouvé.

Indiquez pour chacun de ces algorithmes s'il est correct, en justifiant précisément votre réponse, c'est-à dire en donnant une preuve informelle dans le cas où il est correct, et un scénario contre-exemple dans le cas où il n'est pas correct.

### 1. Algorithme 1

Variable partagée : `global booléen` *`trouvé`*

Code du processus $P$

```
    entier i := 0
(p1) trouvé := faux
(p2) tant que non trouvé faire
(p3)   i++
(p4)   trouvé := (f(i)=0)
(p5) fintq
```

Code du processus $Q$

```
    entier j := 1
(q1) trouvé := faux
(q2) tant que non trouvé faire
(q3)   j - -
(q4)   trouvé := (f(j)=0)
(q5) fintq
```

### 2. Algorithme 2

Variable partagée : `global booléen` *`trouvé`* `:= faux`

Code du processus $P$

```
    entier i := 0
(p1) tant que non trouvé faire
(p2)   i++
(p3)   trouvé := (f(i)=0)
(p4) fintq
```

Code du processus $Q$

```
    entier j := 1
(q1) tant que non trouvé faire
(q2)   j - -
(q3)   trouvé := (f(j)=0)
(q4) fintq
```

### 3. Algorithme 3

Variable partagée : `global booléen` *trouvé* `:= faux`

Code du processus $P$

```
    entier i := 0
(p1) tant que non trouvé faire
(p2)   i++
(p3)   si (f(i)=0) alors
(p4)     trouvé := vrai
(p5) fintq
```

Code du processus $Q$

```
    entier j := 1
(q1) tant que non trouvé faire
(q2)   j - -
(q3)   si (f(j)=0) alors
(q4)     trouvé := vrai
(q5) fintq
```

Pour les deux algorithmes qui suivent, on utilise une instruction **atomique** CAS(x,old,new) (Compare And Swap) qui **en une seule opération**

- renvoie faux si $x \neq old$,
- renvoie vrai si $x = old$, et affecte $new$ à $x$ dans ce cas.

Par ailleurs, l'opération NOP désigne une opération nulle (sans effet).

### 4. Algorithme 4

Variables partagées :

- `global booléen` *trouvé* `:= faux`
- `global entier` *tour* `:= 1`

Code du processus $P$

```
    entier i := 0
(p1) tant que non trouvé faire
(p2)   tant que non CAS(tour,1,2) faire
         NOP
       fintq
(p3)   i++
(p4)   si (f(i)=0) alors
(p5)     trouvé := vrai
(p6) fintq
```

Code du processus $Q$

```
    entier j := 1
(q1) tant que non trouvé faire
(q2)   tant que non CAS(tour,2,1) faire
         NOP
       fintq
(q3)   j - -
(q4)   si (f(j)=0) alors
(q5)     trouvé := vrai
(q6) fintq
```

### 5. Algorithme 5

Variables partagées :

- `global booléen` *trouvé* `:= faux`
- `global entier` *tour* `:= 1`

Code du processus $P$

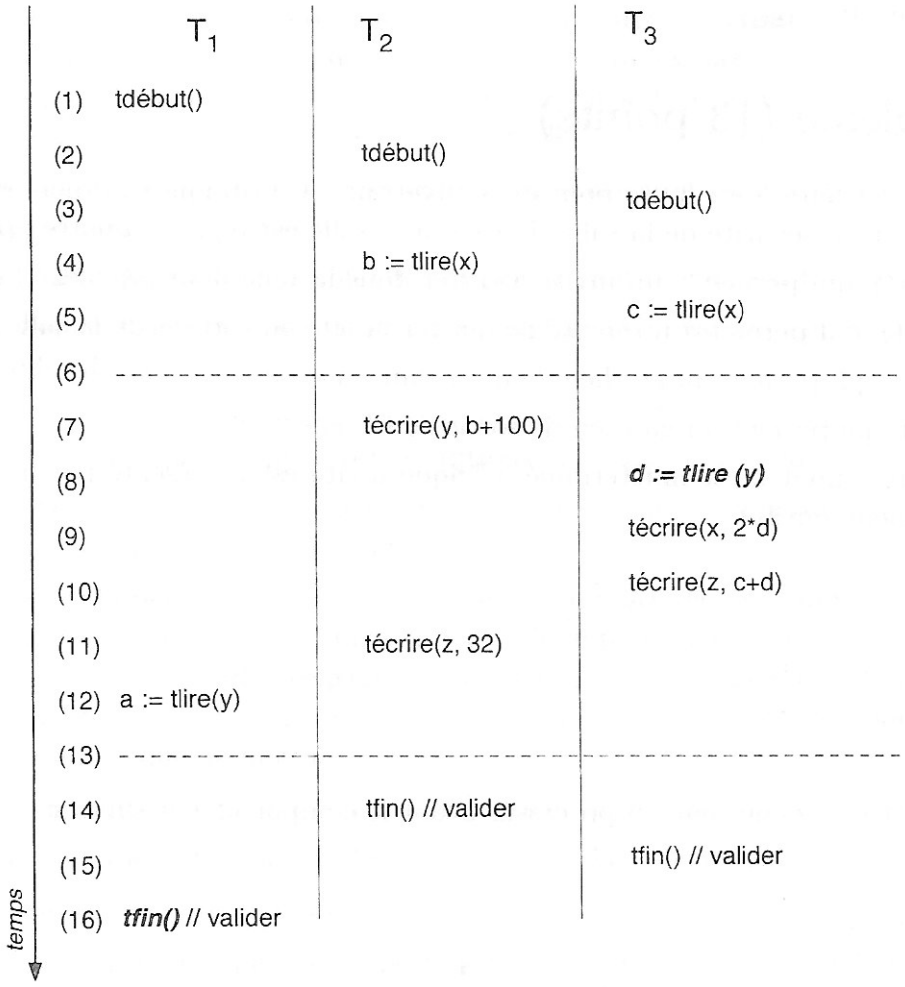
```
    entier i := 0
(p1) tant que non trouvé faire
(p2)   tant que non CAS(tour,1,2) faire
         NOP
       fintq
(p3)   i++
(p4)   si (f(i)=0) alors
(p5)     trouvé := vrai
(p6) fintq
(p7) tour := 2
```

Code du processus $Q$

```
    entier j := 1
(q1) tant que non trouvé faire
(q2)   tant que non CAS(tour,2,1) faire
         NOP
       fintq
(q3)   j - -
(q4)   si (f(j)=0) alors
(q5)     trouvé := vrai
(q6) fintq
(q7) tour := 1
```

## 2 Transactions (6 points)

Le chronogramme suivant représente un scénario d'exécution de trois transactions $T_1$, $T_2$, et $T_3$ opérant sur les variables partagées x, y, et z. Les variables a, b, c ,d sont locales à $T_1$,$T_2$, ou $T_3$. Dans ce chronogramme, le temps réel s'écoule de , et les opérations sont placées par rapport à ce temps réel.




6. Construire le graphe de dépendance correspondant à cette exécution. Précisez pour chacun des arcs l'instant (n) associé à l'opération entraînant la création de l'arc, et indiquer, en justifiant votre réponse, si cette exécution est sérialisable.
7. On suppose maintenant que l'on utilise le protocole de contrôle de concurrence **pessimiste** par estampilles présenté dans le cours. On suppose que l'estampille de la transaction $T_i$ est son indice, soit $i$. On considère que chacune des transactions $T_1$, $T_2$, $T_3$ s'exécute selon le scénario précédent, tant que le protocole de contrôle de concurrence n'intervient pas sur leur exécution. Indiquez quel sera le nouveau scénario obtenu, en justifiant chacune des modifications à l'exécution apportées par le protocole de contrôle de concurrence. Précisez quelles transactions valident et quelles transactions sont abandonnées.
8. On suppose maintenant que l'on utilise le protocole de contrôle de concurrence **pessimiste** par verrouillage à 2 phases présenté dans le cours. On considère que chacune des transactions $T_1$, $T_2$, $T_3$ s'exécute selon le scénario précédent, tant que le protocole de contrôle de concurrence n'intervient pas sur leur exécution. Indiquez quel sera le nouveau scénario obtenu. Faites un bilan de l'état d'exécution des différentes transactions.
9. Même question, pour le scénario où la transaction $T_3$ effectue son opération d :=tlire(y) à l'instant (6) au lieu de l'instant (8).

10. On suppose maintenant que l'on utilise le protocole de contrôle de concurrence optimiste par certification présenté dans le cours, basé sur l'utilisation d'un espace de travail et d'estampilles. Indiquez quel est l'impact de ce protocole sur ce scénario : quelles transactions valident, quelles transactions sont abandonnées ? Précisez pour chacune la raison de l'abandon ou de la validation.
11. Même question, pour le scénario où la transaction $T_3$ termine à l'instant (13) au lieu de l'instant (16).

# 3 Problème (13 points)

Une salle des fêtes a été louée pour un anniversaire. Un (unique) gardien est responsable de l'entretien et de la sécurité de la salle. L'accès à la salle est régi par quatre opérations :

- Accéder(), qui permet à un invité d'entrer dans la salle pour rejoindre la fête en cours ;
- Quitter(), qui permet à un invité de quitter la fête en sortant de la salle ;
- Entrer(), qui permet au gardien d'entrer dans la salle ;
- Sortir(), qui permet au gardien de sortir de la salle ;

Le nombre d'invités est indéterminé. Chaque invité est représenté par un processus ayant le comportement suivant :

```
répéter
Accéder() ;
s'amuser();
Quitter() ;
respirer() ;
sans fin
```

Le gardien est représenté par un processus ayant le comportement suivant :

```
répéter
Entrer() ;
travailler() ;
Sortir() ;
se_reposer() ;
sans fin
```

Les opérations Accéder(), Quitter(), Entrer(), Sortir(), doivent respecter les contraintes suivantes :

- lorsqu'il souhaite entrer, le gardien ne peut accéder à la salle que
  - si elle est vide (pour la préparation ou l'entretien), ou
  - si 200 personnes ou plus se trouvent dans la salle. Dans ce cas, conformément aux dispositions du contrat de location, il impose l'évacuation immédiate de la salle, les normes de sécurité n'étant pas respectées : aucun nouvel invité ne peut entrer, et le gardien reste jusqu'à ce que la salle soit vide de ses invités.
- le gardien ne peut sortir s'il reste des invités dans la salle (situation d'évacuation) ;
- aucun invité ne peut accéder à la salle lorsque le gardien est présent dans la salle (ce qui correspond à la situation où une évacuation est en cours ou bien à la situation où le gardien est en train de préparer la salle)
- un invité peut quitter la salle à tout moment ;
- dès lors que le gardien attend pour entrer ou sortir, et que les conditions sont réunies pour qu'il puisse poursuivre (salle vide ou suroccupée), il doit être débloqué et progresser, avant que toute autre opération soit entamée ou reprise.

Le nombre d'invités peut varier, et n'est pas borné, mais l'on suppose (pour simplifier) que la capacité physique de la salle (supérieure à la capacité réglementaire de 200 personnes) est toujours suffisante pour qu'ils y trouvent un espace.

Donner le code des quatre opérations Accéder(), Quitter(), Entrer(), et Sortir(), de manière à assurer les contraintes précédentes (et à garantir que dès lors qu'une ou plusieurs opérations sont possibles au regard des contraintes, au moins l'une de ces opérations est exécutée) :

12. (4 points) En supposant que l'on dispose de **moniteurs de Hoare** (exécution automatique en exclusion mutuelle des opérations, priorité au signalé, files FIFO associées aux variables condition). On réalisera le moniteur en suivant la démarche vue en TD, basée sur l'évaluation de conditions d'état pour contrôler la progression des processus.
    (a) (1 point) Ecrire (en français) les conditions d'acceptation des opérations Accéder(), Quitter(), Entrer(), et Sortir().
    (b) (0,5 point) Définir les variables d'état permettant de représenter ces conditions.
    (c) (0,5 point) Enoncer ces conditions et établir un invariant du moniteur liant ces variables d'état
    (d) (2 points) Programmer les opérations du moniteur. Préciser (sous forme de commentaire dans le code) les pré/post conditions des signaler/attendre.
13. (2 points) En construisant une tâche serveur Ada, en suivant une méthodologie au choix (conditions d'acceptation ou automate)
14. (2 points) Sans faire d'hypothèses sur la fréquence des arrivées, on suppose néanmoins que le flux d'arrivées d'invités ne se tarit jamais. On suppose par ailleurs que les opérations « applicatives » s'amuser(), respirer(), travailler(), et se_reposer() se terminent toutes, toujours au bout d'un temps fini. Dans ces conditions, le protocole d'accès à la salle présente-t-il un risque de famine
    — pour les invités, dans le cas des moniteurs et dans le cas des tâches Ada ?
    — pour le gardien, dans le cas des moniteurs et dans le cas des tâches Ada ?

    Justifiez précisément vos réponses.
15. (5 points) En supposant que l'on dispose de sémaphores (généraux), définis par une classe Sémaphore, fournissant **uniquement** les méthodes P(), V(), et un constructeur permettant de fixer la valeur initiale du sémaphore.
    *Indication :* Il pourra être utile de représenter, pour le gardien (par des booléens ou un type énuméré) un état présent (dans la salle) et un état d'attente (pour entrer ou sortir)
    *Précisions sur le barème :* ce problème est sensiblement plus difficile à résoudre avec les sémaphores. Les points seront attribués pour une part en fonction de la pertinence des schémas employés par votre solution, et de votre analyse de ses faiblesses éventuelles ; et pour l'autre part, en fonction de la proximité de votre solution avec une solution correcte.