

# Université Paris Diderot – Master 1 II

## Théorie et pratique de la concurrence

Partiel du 30 avril 2009

Durée : 1h30. Tous les documents sont autorisés. Le barème est indicatif.

### Question 1 :

(7 points)

Soit le programme Promela suivant :

```
int n=0;
active proctype Q() {
q1: n=n+1;
q2: n=n+1;
q3: skip
}

active proctype P() {
p1: do :: n < 2 ->
p2:      print n
      :: else ->
p3:      print '␣'
      od
}
```

1. Construire les exécutions qui affichent les mots suivants (donnés sous forme d'expressions régulières) :  $012^*002^*0^*$ .
2. Ecrire en LTL les propriétés suivantes en explicitant les proposition atomiques utilisés :
  - (a) “2 doit apparaître dans l’affichage”
  - (b) “2 apparaît au plus une fois”
  - (c) “2 apparaît une infinité de fois”
  - (d) “2 peut apparaître que après 0”
3. Pour chaque propriété ci-dessus, indiquer sa classe (vivacité, sûreté ou les deux) et sa valeur de vérité pour le programme considéré.

### Question 2 :

(8 points)

L’algorithme du “jeton circulaire” (*token ring*) assure l’exclusion mutuelle entre de processus distribués, communiquant par échange de messages. Soient  $N \geq 2$  sites connectés en anneau (le site  $i$  communique uniquement avec ses “voisins”, les sites  $i - 1 \bmod N$  et  $i + 1 \bmod N$ ) par des canaux de taille 1. Sur chaque site, il existe un processus qui doit exécuter (de façon répétée) un travail en exclusion mutuelle par rapport aux processus travailleurs des autres sites. Le principe de cet algorithme est de faire circuler dans l’anneau un “jeton” unique. Le site qui a le jeton autorise son processus travailleur à effectuer son travail. Un fois le travail fini, le site communique le jeton à son voisin  $i + 1 \bmod N$  et ainsi de suite.

1. Donner une modélisation en Promela pour cet algorithme en explicitant
  - (a) le type des messages échangés sur les canaux et la déclaration des canaux,
  - (b) les processus participant à cet algorithme et leur appartenance aux sites.
2. Exprimer en LTL la propriété d’exclusion mutuelle à tester pour votre modélisation.
3. Commenter pour cet algorithme la propriété d’attente bornée.

**Question 3 :**

(7 points)

Soit l'algorithme d'exclusion mutuelle suivant modélisé en Promela :

```

1 #define N 5
2 int requete = 0;
3 int reponse = 0;
4 active proctype Serveur () {
5     do :: true ->
6         (requete != 0);
7         reponse = requete;
8         (reponse == 0);
9         requete = 0
10    od
11 }

1 active [N] proctype Client () {
2     do :: true ->
3         /* section non critique */
4         do :: reponse != _pid ->
5             requete = _pid
6             :: else -> break
7         od;
8         /* section critique */
9         reponse = 0
10    od
11 }
```

1. Expliquer (1 page maximum) comment cet algorithme fonctionne. Rappel : `_pid` est le numéro unique associé à chaque processus (le processus `Serveur` ayant le `_pid` 0).
2. Montrer, en raisonnant sur l'algorithme, que cet algorithme satisfait les propriétés d'exclusion mutuelle et d'absence d'inter-blocage.
3. Qu'en est-il des propriétés d'absence de famine et d'attente bornée ?
4. Y-a-t-il un processus privilégié parmi les clients ?

**Question 4 :**

(8 points)

Dans une ville tranquille, un coiffeur possède un petit salon ayant une porte d'entrée, une porte de sortie, un fauteuil de coiffure et  $N$  chaises. Les clients arrivent par la porte d'entrée et sortent par la porte de sortie après avoir eu leur coupe de cheveux. Comme le salon est petit, uniquement le client sur le fauteuil de coiffure peut être servi à un moment donné par le coiffeur. Le coiffeur passe sa vie entre dormir et servir ses clients. Quand il n'a aucun client, le coiffeur dort. Quand un client arrive et le fauteuil est libre, il s'assoit dans le fauteuil et réveille le coiffeur. Si le fauteuil n'est pas libre, le client occupe une chaise s'il y a des chaises libres ou il attend qu'une chaise se libère sinon. Un client sur une chaise attend que le fauteuil se libère. Après avoir fini une coupe, le coiffeur fait sortir le client servi et s'endort.

Modéliser ce problème en utilisant les sémaphores pour la synchronisation entre le coiffeur et ses clients. Pour cela, écrire (dans le langage de votre choix) :

1. un processus générique `Client` dans lequel sont identifiés clairement ses états : en attente d'une chaise, sur une chaise en attente du fauteuil, sur le fauteuil en attente de la fin de sa coupe et servi,
2. un processus `Coiffeur` avec les états : endormi et en service et
3. des déclarations de sémaphores avec leur type (binaire ou généralisé) et leur valeur initiale.
4. De plus, en utilisant LTL, spécifier la propriété suivante : "le coiffeur est réveillé que si le fauteuil est occupé".

**Correction 1 :**

(1 + 4 + 2)

FIG. 1 – Point 1.1 : séquences d'exécution.

**Point 1.2 :** Pour capturer le fait que une certaine valeur  $v$  de  $n$  est imprimée pendant l'exécution, on définit le prédicat suivant :

$$print(v) = (n = v) \wedge P@p2 \wedge \bigcirc P@p1$$

En instanciant  $v$  par 0 et 2, on obtient les deux prédicats  $print0$  et  $print2$  utilisés dans les formules ci-dessous.

1. a.  $\diamond print2$
2. b.  $\square \neg print2 \vee \square (print2 \Rightarrow \bigcirc \square \neg print2)$
3. c.  $\square \diamond print2$
4. d. Propriété difficile, car on parle du passé et qu'on ne demande pas que 0 et 2 se suivent ! On décompose la propriété en deux cas disjoints (c'est donc une disjonction de deux formules) :
  - (a) si 0 n'apparaît pas, alors 2 n'apparaît pas non plus :  $(\neg \diamond print0) \Rightarrow (\neg \diamond print2)$
  - (b) il n'est pas possible d'avoir un 2 avant la première occurrence de 0 (ici 0 est forcé d'apparaître) :  $(\neg print0 \vee \neg print2) \mathcal{U} print0$

**Point 1.3 :**

1. a. Propriété de vivacité, fausse (voir séquence  $0^* 1^*$ ).
2. b. Propriété de sûreté (2 ne doit pas apparaître 2 fois), vraie car une fois que 2 est affiché, le processus  $P$  choisit que la branche  $\leq \sim$  du test, donc il n'imprime plus  $n$ .
3. c. Propriété de vivacité, fausse car complémentaire de b.
4. d. Propriété de sûreté (2 ne doit apparaître avant un 0), fausse (voir séquence  $(q1, p1, 0) \rightarrow (q1, p2, 0) \rightarrow (q2, p2, 1) \rightarrow (q3, p2, 2) \xrightarrow{print2} (q3, p1, 2) \dots$ ).

**Correction 2 :**

(1 + 4 + 2 + 1)

Comme dans les algorithmes d'exclusion mutuelle dans un cadre distribué, sur chaque site il faut faire fonctionner deux processus : un qui effectue le travail et demande la section critique (**Worker** ci-dessous) et un autre qui gère les messages (**Msg** ci-dessous). Ces deux processus communiquent entre eux par une mémoire partagée sur chaque site (donc c'est chaque variable est un tableau de taille  $N$ ) : un booléen **requete** indique une demande de SC de la part du travailleur et un booléen **reponse** indique si la requette est satisfaite.

**Point 2.1a :** Comme convention, les canaux relient les processus tel que le processus  $i$  lit son canal et envoie un message sur le canal  $i + 1 \% N$ . Le message échangé est vide, mais pour simplifier, on considère qu'il s'agit d'un booléen.

```
1 chan ch[N] = [1] of bit ;
```

**Point 2.1b :** Le code exécuté par chaque composant sur chaque site est :

```
1 bit requete[N] = 0;
2 bit reponse[N] = 0;

1 active [N] proctype Worker() {
2   do :: 1 ->
3     /* section non-critique */
4     skip;
5     /* pre-protocole */
6     requete[_pid] = 1;
7     (reponse[_pid] == 1);
8 sc: /* section critique */
9     skip;
10    /* post-protocole */
11    reponse[_pid] = 0;
12    requete[_pid] = 0;
13  od
14 }

1 active [N] proctype Msg() {
2   int id = _pid - N;
3   bit b;
4   do :: 1 ->
5     ch[id] ? b;
6     if :: requete[id] == 1 ->
7         reponse[id] = 1;
8         (requete[id] == 0)
9         :: else -> skip
10    fi;
11    ch[id+1 % N] ! b
12  od
13 }
```

On suppose qu'initialement, le canal d'index 0 contient un message.

**Point 2.2 :** Si on définit par  $sc_i = Worker[i]@sc$  alors l'exclusion mutuelle s'écrit en LTL :  
 $\Box \neg \bigwedge_{i \neq j} sc_i \wedge sc_j$ .

**Point 2.3 :** Pour la modélisation ci-dessus, la propriété d'attente bornée est satisfaite car, si un processus fait la demande de la SC, sa demande sera satisfaite après au plus  $N - 1$  exécutions de SC. En effet, le jeton est bloqué sur un site uniquement si le **Worker** exécute la SC, sinon, le jeton est transmis au site suivant (c'est pourquoi on a besoin de processus **Msg**). Comme les SC se terminent forcément, le temps d'attente du demandeur est donc borné.

**Correction 3 :** (2 + 2 + 1 + 1 + 1)

**Point 3.1 :** Il s'agit d'un algorithme d'exclusion mutuelle centralisé dans un système à mémoire partagée : un serveur reçoit des requêtes de la part des processus accédant à la SC à travers la variable **requete**. Cette requête contient le numéro *unique, strictement positif*, du demandeur. Le serveur répond en affectant la variable **reponse** à un numéro reçu (le dernier vu dans **requete** avant l'affectation). Le processus demandeur ayant ce numéro peut ainsi entrer dans sa section critique. Le post-protocole consiste à changer la valeur de la **reponse** à 0 afin d'indiquer au serveur qu'il peut traiter un autre demandeur.

**Point 3.2 :**

– Exclusion mutuelle : les invariants utilisés pour la preuve sont :

1. Tous les clients ont des pids différents et strictement positifs :
2. Quand le serveur se trouve en dehors de la ligne 8, la variable **reponse** est 0 :

$$(\forall \ell \in \{5,6,7,9\} \text{Server}@ \ell) \Rightarrow (\text{reponse} = 0)$$

3. La variable **reponse** est 0 alors tous les clients sont dans la section non-critique ou dans le pre-protocole :

$$(\text{reponse} = 0) \Rightarrow (\wedge_{j \in [0, N-1]} \vee_{m \in [2,5]} \text{Client}[j]@m)$$

4. Quand la variable **reponse** est non nulle, le serveur est en attente à la ligne 8 :

$$(\text{reponse} \neq 0) \Rightarrow \text{Server}@8$$

5. Quand la variable **reponse** est non nulle, elle a des valeurs parmi les pids des clients :

$$(\text{reponse} \neq 0) \Rightarrow 1 \leq \text{reponse} \leq N$$

(facile à vérifier en regardant les endroits où les variables du programme sont affectées).

6. De plus, un client peut se trouver aux lignes 6–9 :

$$(\text{reponse} \neq 0) \Rightarrow \exists j. 1 \leq j \leq N \wedge \vee_{m \in [6,9]} \text{Client}[j]@m$$

Ces invariants sont faciles à vérifier sur le code.

Le code des clients montre que un client arrive en SC que si, à un certain moment, la variable **reponse** a été égale à son pid, donc non nulle. Dans ce cas, le serveur est bloquée à la ligne 8 et ne peut pas changer la valeur de **reponse** (à la ligne 7). De plus, les autres clients sont bloqués aux lignes 2–5. Donc au plus un client se trouve en SC tant que **reponse** a une valeur non-nulle.

- Absence d'interblocage : si plusieurs clients exécutent en même le pré-protocole, le dernier qui affecte la variable **requete** avant que le serveur effectue l'affectation de la ligne 7 sera choisi par le serveur.

**Point 3.3 :** La famine est possible, donc la propriété d'attente bornée n'est pas satisfaite. En effet, en prenant comme exemple le cas de deux clients, il est possible d'exhiber une trace dans laquelle le premier client est en famine (on choisit comme états l'état du serveur, celui des deux clients et les valeurs de **requete** et **reponse**) :  $\dots \rightarrow (S@6, C[1]@4, C[2]@4, 0, 0) \rightarrow (S@6, C[1]@5, C[2]@4, 0, 0) \rightarrow (S@6, C[1]@5, C[2]@5, 0, 0) \rightarrow (S@6, C[1]@4, C[2]@5, 1, 0) \rightarrow (S@6, C[1]@4, C[2]@4, 2, 0) \rightarrow (S@7, C[1]@4, C[2]@4, 2, 0) \rightarrow (S@8, C[1]@4, C[2]@4, 2, 2) \rightarrow (S@8, C[1]@4, C[2]@7, 2, 2) \rightarrow (S@8, C[1]@4, C[2]@8, 2, 2) \rightarrow (S@8, C[1]@4, C[2]@9, 2, 2) \rightarrow (S@8, C[1]@4, C[2]@4, 2, 0) \rightarrow (S@9, C[1]@4, C[2]@4, 2, 0) \rightarrow (S@6, C[1]@4, C[2]@4, 0, 0) \rightarrow \dots$

**Point 3.4 :** Il n'y a pas de client privilégié car :

- le serveur ne sélectionne pas un numéro spécial de requête et
- l'affectation de la variable **requete** dans le clients ne tiens pas compte d'une valeur spécifique.

**Correction 4 :** ()

**Point 4.3 :** les sémaphores utilisés sont :

- **veille** : un sémaphore binaire (initialisé à 0) pour le barbier (initialement endormi),
- **fincoupe** : un sémaphore binaire (initialisé à 0) pour le client servi (initialement aucun, c'est le barbier qui le signale et le client qui se bloque en attente de de fin service),
- **faut-libre** : sémaphore binaire (initialement à 1) pour le fauteuil,
- **chaise-libre** : sémaphore généralisé (initialement à  $N$ ) pour les chaises.

```

1 bsem veille = 0;
2 bsem fin-coupe = 0;
3 bsem faut-libre = 0;
4 sem chaise-libre = N;

```

**Point 4.1 :** le code du client

```

1 active [M] proctype Client() {
2   do :: 1 ->
3     /* vie normale */ skip;
4     /* coiffeur */
5     wait(chaise-libre);
6     wait(faut-libre);
7     signal(chaise-libre);
8     signal(veille);
9     wait(fin-coupe);
10    signal(faut-libre)
11  od
12 }

```

**Point 4.2 :** on a considéré que le coiffeur essaie de s'endormir entre deux clients :

```

1 active proctype Coiffeur() {
2   do :: 1 ->
3     /* dors */ skip;
4     wait(reveil);
5     /* travaille */ skip;
6     signal(fin-coupe)
7   od
8 }

```

**Point 4.4 :** Le coiffeur est réveillé aux lignes 5-6 ( $cr = \text{Coiffeur@5-6}$ ) et le fauteuil est occupé quand le sémaphore `faut-libre` est 0 ( $fo = \text{faut-libre} = 0$ ). Alors la propriété s'écrit :  $\square[fo \Rightarrow cr]$ .