NIGET Tom
Note: 16/20 (score total : 53.5/66)

+24/1/42+

UNIVERSITÉ NICE SOPHIA ANTIPOLIS — POLYTECH'NICE SOPHIA

Année 2021-22
Programmation concurrente
QCM n°2

Nom et prénom : NIGET Tom

Vous devez obligatoirement répondre en noicissant les cases sans utiliser le blanc masque. Certaines questions n'ont peut être pas de bonnes réponses, d'autres une ou plusieurs.

Barême :

- *Questions fermées simples :* +3 bonne réponse, 0 pas de réponse, -0.5 mauvaise réponse ou plus d'une case cochée
- *Questions fermées multiples (♣) :* +1 bonne case cochée ou mauvaise case non cochée, 0 pas de réponse, -0.5 bonne case non cochée ou mauvaise case cochée, -1 minimum possible
- *Questions ouvertes :* le barême est indiqué dans le cartouche

Acquisition simultanée de ressources

On dispose d'un ensemble de ressources notées R1, ..., Rn, manipulées par des processus concurrents à l'aide de la procédure Opération (Ri) qui effectue un traitement sur la ressource Ri.

Dans cet exercice on s'intéresse valider le code de la procédure OpérationComposée (Ri, Rj) qui effectue un traitement sur les ressources Ri et Rj..

Dans la suite de l'exercice, m, S1, ..., Sn désigne des sémaphore initialisés à 1.

Pour chacun des propositions suivantes, vous demandez étudier les 3 propriétés suivantes :

- Est-ce que la proposition de mise en oeuvre garanti la propriété d'atomicité ?
- Y-a-t-il un risque d'interblocage ?
- st-ce que la proposition de mise en oeuvre permet une exécution en parallèle si elle porte sur deux paires de ressources différentes.

Proposition : m.down; Opération(Ri); Opération(Rj); m.up

Question 1

3/3 ☐ pas atomicité ☒ atomicité

Question 2

3/3 ☒ pas interblocage ☐ interblocage

Question 3

3/3 ☐ parallélisme ☒ pas parallélisme

Proposition : Si.down; Sj.dow; Opération(Ri); Opération(Rj); Sj.up; Si.up

Question 4

3/3 ☐ pas atomicité ☒ atomicité

Question 5

-0.5/3 ☒ interblocage ☒ pas interblocage

Question 6

3/3 ☐ pas parallélisme ☒ parallélisme

Allocation multiples

Plusieurs processus Pi se partagent une ressource commune existant en N exemplaires. Le code de chaque processus consiste en une succession de demande de ressources et se termine par une succession de libération de l'ensemble des ressources réservées.

Chaque processus Pi demandera au maximum Pi-max ressources et les ressources acquises par un processus sont indisponibles pour les autres jusqu'à ce qu'elles soient libérées (accès exclusif). Par conséquent, si les ressources qu'un processus demande sont en nombre insuffisantes, sa demande n'est pas satisfaite, aucune ressource ne lui est allouée et il se bloque. Il sera réveillé et recevra ses ressources quand l'allocation deviendra possible.

Question 7 Le code suivant est une proposition de mise en oeuvre :

```
            // variables globales partagées
            1 mutex    : sémaphore  init 1;
            2 attente  : sémaphore  init 0;
            3 n_libre  : entier  init N;      // nombre de ressources disponibles
            4 n_att    : entier  init 0;      // nombre de processus en attente
5 Procédure demander (n) {          14 Procédure libérer (n) {
6     mutex.down();                 15     mutex.down();
7     tantque (n>n_libre) {         16     n_libre:=n_libre+n;
8         n_att := n_att+1;         17     tantque (n_att>0) {
9         attente.down();           18         attente.up();
10    }                             19         n_att := n_att-1;
11    n_libre:=n_libre - n;         20     }
12    mutex.up();                   21     mutex.up();
13 }                                22 }
```

Est-ce que le code précédemment donné initialise correctement les deux sémaphores .

3/3

☒ oui ☐ non

Question 8 En supposant les sémaphores correctement initialisés, est-ce que la procédure libérer(n) est correcte ?

3/3

☐ non ☒ oui

Question 9 Corrigez éventuellement la procédure précédente et donnez les valeurs correctes d'initialisation des sémaphores. Si rien n'est à corriger, précisez-le explicitement sur votre copie.

☐0 ☐1 ☐2 ☒4

4/4

Je pense que le code est correct le mutex est bien à 1, l'attente est bien un sémaphore privé, ~~les deux fonctions~~ me semblent correctes. demander encore

Question 10 En supposant que le code des procédures demandés et libérés soient correct. Est-ce que le scénario suivant présente un risque d'interblocage ?

- Nombre de ressources fixés à 19
- Demandes successives du processus 1 : 2, 2, 2
- Demandes successives du processus 2 : 1, 2, 3
- Demandes successives du processus 3 : 3, 3, 3

3/3

☒ non ☐ oui

Question 11 On souhaite implémenter l'allocateur sous la forme d'un moniteur. Définissez les variables du moniteur et leurs valeurs d'initialisation. ☐0 ☐1 ☐2 ▨4

4/4

```
int libre = 19;
```

Question 12 Ecrivez la procédure demander sous la forme d'un moniteur. ☐0 ☐1 ☐2 ☐4 ▨6

6/6

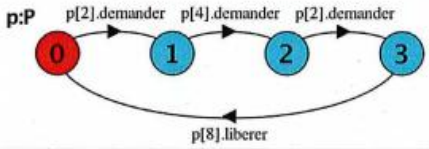
```
synchronized void demander (int N)
{
while (libre < N) wait();
libre -= N;
notifyAll();
}
```

On souhaite par ailleurs, modéliser le même problème en FSP. Nous proposons la modélisation correcte suivante :

```
const N=15          // nombre d'exemplaire disponible
RESSOURCE = RESSOURCE[N],
RESSOURCE[n:0..N] = (demander[i:0..n] -> RESSOURCE[n-i]
                    |liberer[i:0..N-n] -> RESSOURCE[n+i]).

// le processus P demande 2 exemplaires de la ressource, puis 4 et puis 2.
P = (demander[2] -> demander[4] -> demander[2]
      // le processus P rend toute ses ressources en une seule fois
      -> liberer[8] -> P)+{demander[i:0..N], liberer[i:0..N]}.

Q = (demander[4] -> demander[2] -> demander[3]
       -> liberer[9] -> Q)+{demander[i:0..N], liberer[i:0..N]}.
```

```
R = (demander[3] -> demander[3] -> demander[3]
      -> liberer[9] -> R)+{demander[i:0..N], liberer[i:0..N]}.

||APPLICATION = ({p,q,r}::RESSOURCE || p:P || q:Q || r:R).
```

Voici par exemple, le processus p:P en LTS :



Question 13 Est-ce que dans la modélisation FSP, une série de demandes de ressources insuffisantes se détectera sous la forme d'un interblocage ?

-0.5/3

property VIV = (f[n] f[r]demander

☒ oui ☒ non

Question 14 Excrivez en FSP la propriété de vivacité VIV garantissant que le processus P puisse toujours demander des ressources (absence de famine).

☐0 ☐1 ☒2 ☐4

2/4

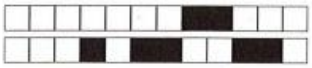
property VIV = (demander -> demander)

Question 15 Soit VIV une propriété de vivacité. Que faut-il faire au niveau du processus ||APPLICATION et/ou dans l'outil LTSA pour vérifier que la propriété est vérifiée ?

☐0 ☐1 ☐2 ☒4

4/4

Il faut mettre le processus en parallèle avec la propriété
APPLICATION || VIV
Et il y a une fonction "Check" dans LTSA.



Question 16
Combien de ligne faut-il modifier au minimum pour avoir une tranformation FSP -> Java correcte ?

```
BRIDGE = BRIDGE[0],
BRIDGE[n:T] = (when(n==0) enter -> BRIDGE[n+1]
              |exit -> BRIDGE[n-1]).

// ne pas prendre en compte la non declaration des exceptions
class Bridge {
  private int n  = 1;
  synchronized void enter() {
    while (n!=0) wait();
    n++;
    notifyAll();
  }
  synchronized void exit() {
    --n;
   notifyAll();
  }
}
```

3/3

☐ 2 ☐ 0 ☐ 3 ☒ 1

Question 17 Dans l'algorithme du banquier, est-ce que l'état suivant est sain ?

D	Util	Max
P1	6	10
P2	6	9
P3	3	8
P4	3	7
Libre : 3		

-0.5/3

☒ non ☒ oui

Aujourd'hui c'est jour d'entrainement

Un stade d'athlétisme peut recevoir les athlètes de trois clubs A, B et C qui viennent s'y entraîner. Pour organiser les entrainements, à un instant donné, la ou les règles suivantes sont imposées :

- le stade ne peut recevoir que des athlètes d'un même club,

Nous modélisons en FSP, ce problème de la manière suivante :

```
const C = 2
const N = 5
const M = 10


ACCES_STADE = ACCES_STADE[0][0][0],
ACCES_STADE[a:0..M][b:0..M][c:0..M] = (when (a<M) a.entre -> ACCES_STADE[a+1][0][0]
                        | when (b<M) b.entre -> ACCES_STADE[0][b+1][0]
                        | when (c<M) c.entre -> ACCES_STADE[0][0][c+1]
                        | when (a>0) a.sort -> ACCES_STADE[a-1][0][0]
                        | when (b>0) b.sort -> ACCES_STADE[0][b-1][0]
                        | when (c>0) c.sort -> ACCES_STADE[0][0][c-1]).

E_JOUEUR = (entre -> E_JOUEUR).
```

```
S_JOUEUR = (sort -> S_JOUEUR).

||STADE = ({a,b,c}:E_JOUEUR || {a,b,c}:S_JOUEUR || ACCES_STADE).
```

Question 18 Modifiez la ligne : "when (c<M) c.entre -> ACCES_STADE[0][0][c+1]" pour que la ou les propriétés souhaitées soient respectées. ☐0 ☐1 ☐2 ☒4

4/4

when (c<M and a==0 and b==0) c.entre -> - - - - -

Question 19 Proposez une fonction du moniteur qui implémente la règle FSP que vous avez précédemment écrites. ☐0 ☐1 ☐2 ☒4

4/4

```
int a=0, b=0, c=0;
- - -
synchronized void cmentre()
{ while(c>=M || a!=0 || b!=0) wait();
  c++;
  notifyAll();
}
```

int a, b, c) :0

sync void c.entre()
while (c>=1 || b!=0 || a!=0