NIGET Tom
Note: 15 / 20
(score total : 47 / 63)

+39/1/45+

UNIVERSITÉ NICE SOPHIA ANTIPOLIS POLYTECH'NICE SOPHIA

Année 2021-22
Programmation concurrente
QCM n°1

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

Question 1 Est-ce que ces deux opérations read et write ci-dessous, regroupées dans la même classe Disk sont correctement programmées ?

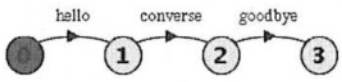
```
// Les opérations seek, read et write s'exécutent en exclusion mutuelle
int disk_read (sector x) {
        int r;
        D.seek(x);
        r := D.read();
        return (r);
}

void disk_write (sector x, int v) {
     D.seek(x);
     D.write(v);
}
```

compétition

☒ non ☐ oui

3 / 3

Question 2 ♣ Quel est le processus FSP correspondant à la description LTS suivante :



- ☐ `P=(hello -> converse -> goodbye).`
- ☒ `MEETING=(hello -> converse -> goodbye -> STOP).`
- ☒ `P=(hello -> converse -> goodbye -> STOP).`
- ☐ `MEETING=(hello -> converse -> goodbye).`
- ☐ `MEETING=(hello | converse | goodbye).`
- ☐ `MEETING=(hello -> converse -> goodbye -> MEETING).`
- ☐ *Aucune de ces réponses n'est correcte.*

6 / 6

Question 3 Soient trois processus concurrents P1, P2 et P3 qui partagent les variables n et out. Pour contrôler les accès aux variables partagées, un programmeur propose les codes suivants :

```
Semaphore mutex1 = 1 ; Semaphore mutex2 = 1 ;

Code du processus p1 : {
mutex1.down() ; mutex2.down() ;
out=out+1 ; n=n-1 ;
mutex2.up(), mutex1.up() ;
}

Code du processus p2 : {
mutex2.down(); out=out-1 ; mutex2.up() ;
}

Code du processus p3 : {
mutex1.down() ; n=n+1 ; mutex1.up() ;
}
```

Si cette proposition vous semble incorrecte, indiquez pourquoi, sinon laisser la case vide.

☒0 ☐1 ☐2 ☐3 ☐4 0 / 4

Semaphore mutex1 = new Semaphore (1);
Il mutex2 (2);

Question 4 ♣ Quelle sont les conditions nécessaires pour avoir des données incohérentes :

- ☐ Une ressource partagée en lecture
- ☒ Plusieurs threads
- ☐ Une ressource exclusive
- ☐ Une seule thread
- ☒ Une ressource partagée en lecture et écriture
- ☐ *Aucune de ces réponses n'est correcte.*

5 / 5

Question 5 ♣ En programmation concurrente, qu'est-ce qu'un MUTEX et qu'elle est son utilisation ?

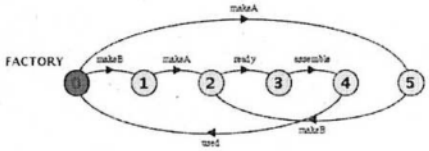
- [x] Un semaphore initialisé à 1
- [x] A contrôler l'usage d'un ressource partagée
- [x] A proteger une section critique
- [ ] Un semaphore initialisé à 0
- [ ] *Aucune de ces réponses n'est correcte.*

4 / 4

Question 6 Soit le programme FSP suivant :

```
MAKE_A    = (makeA->ready->used->MAKE_A).
MAKE_B    = (makeB->ready->used->MAKE_B).
ASSEMBLE = (ready->assemble->used->ASSEMBLE).

||FACTORY = (MAKE_A || MAKE_B || ASSEMBLE).
```

Est-ce que le diagramme LTS suivant correspond au processus ||FACTORY ?



- [x] oui
- [ ] non

3 / 3

Question 7 Est-ce que les processus S1 et S2 ont le même comportement ?

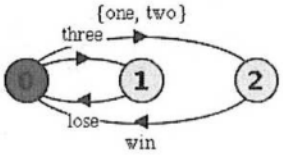
```
P = (a -> b -> P).
Q = (c -> b -> Q).
||S1 = (P || Q).

S2 = (a -> c -> b -> S2 | c -> a -> b -> S2).
```

graphe différent
mais comportement identique

- [ ] non
- [x] oui

3 / 3

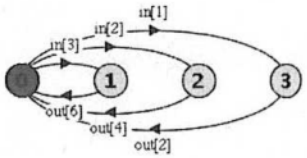
Question 8 ♣ Quel est le processus FSP correspondant à la description LTS suivante :



- [ ] `GAME=(one -> p | two -> p | three -> q), p=(win -> GAME), q=(lose -> GAME).`
- [x] `GAME=({one, two} -> win -> GAME | three -> lose -> GAME).`
- [x] `GAME=(one -> P | two -> P | three -> Q), P=(win -> GAME), Q=(lose -> GAME).`
- [ ] `GAME=(one -> P | two -> P | three -> Q). P=(win -> GAME). Q=(lose -> GAME).`
- [x] `GAME=(one -> win -> GAME | two -> win -> GAME | three -> lose -> GAME).`
- [ ] *Aucune de ces réponses n'est correcte.*

2 / 5

Question 9 ♣ Quel est le processus FSP correspondant à la description LTS suivante (DOUBLE) :

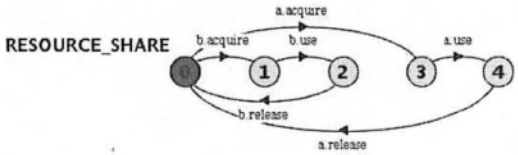


- [x] `DOUBLE=(in [i:1..3] -> DOUBLE[i]), DOUBLE[j:1..3]=(out [2*j] -> DOUBLE).`
- [x] `DOUBLE=(in [i:1..3] -> out [2*i] -> DOUBLE).`
- [x] `DOUBLE(I=3)=(in [i:1..I] -> out [2*i] -> DOUBLE).`
- [ ] *Aucune de ces réponses n'est correcte.*

3 / 3

Question 10 Quel est le processus le processus LTS correspondant au diagramme FSP suivant :

```
USER=(acquire -> use -> release -> USER).
RESOURCE=(acquire -> release -> RESOURCE).
```



- [ ] `||RESOURCE_SHARE=({a,b}:USER || {a,b}:RESOURCE).`
- [ ] `||RESOURCE_SHARE=({a,b}::USER || {a,b}:RESOURCE).`
- [ ] `||RESOURCE_SHARE=({a,b}::USER || {a,b}::RESOURCE).`
- [x] `||RESOURCE_SHARE=({a,b}:USER || {a,b}::RESOURCE).`

3 / 3

Question 11 Est-ce que le code Java suivant implémente bien l'accès à une section critique ?

```
// avant section critique
synchronized(this) {
// je suis en section critique
}
// après section critique
```

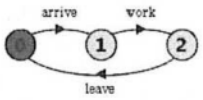
- [ ] non
- [x] oui

3 / 3

Question 12 Donnez la définition de la propriété de sureté. ☒0 ☐1 ☐2 ☐4 0 / 4

Un programme est dit sûr s'il ne peut se retrouver en situation de blocage, c'est-à-dire que le programme boucle à l'infini ou bien s'arrête sur un état terminal mais ne s'arrête pas sur un état quelconque

Question 13 ♣ Quel est le processus FSP correspondant à la description LTS suivante :



- ☐ `JOB=(arrive -> work -> leave -> job).`
- ☒ `JOB=(arrive -> work -> leave -> JOB).`
- ☐ `JOB=(arrive | work | leave | JOB).`
- ☒ `P=(arrive -> work -> leave -> P).`
- ☐ `JOB=(arrive -> work -> leave).`
- ☐ *Aucune de ces réponses n'est correcte.*

5 / 5

Question 14 Comment détecte-t-on sur un diagrame LTS qu'un programme est sur. ☐0 ☐1 ☒2 ☐4 2 / 4

Sur un diagramme LTS, il y a toujours au moins une transition depuis un état intermédiaire vers un autre état (intermédiaire ou autre processus).

ERROR / STOP?

END

Question 15 Soit le programme Java suivant utilisé dans une application :

```
class Compte {
   private int solde = 100;
   public int getSolde()  { return solde; }
   public void retirer (int montant) { solde -= montant; }
   public void deposer (int montant) { solde += montant; }
}
```

On souhaite réutiliser ce code mais maintenant, deux threads souhaitent partager un même objet instance de cette classe. Si vous pensez qu'il faut modifier le code de la classe 'Compte', proposer la nouvelle version de la classe ci-dessous ; sinon laisser la case vide. ☐0 ☐1 ☐2 ☐3 ☒4 4 / 4

private volatile int solde = 100;

public synchronized void retirer (...)

/ - - - - , , deposer (...)

// Pas la peine de synchronized pour getSolde car lecture seulement.

Question 16 Décrivez le fonctionnement d'un verrou et ses principales primitives. ☐0 ☒1 ☐2 ☐4 1 / 4

Un verrou est une ressource pouvant être acquise par au plus 1 thread à la fois, et possède les primitives

- acquire : acquérir le verrou (exclusivement)

- release : libérer le verrou

acquire/release opérations atomiques s'exécutant en exclusion mutuelle