Université Ferhat Abbas Sétif 1 **Faculté des Sciences** **Département d'informatique** **Examen SR – M1 – 2024 – 01h30**	Nom : ..... Prénom : ..... Spécialité : ..... Groupe : .....

Questions de Cours (10 pts) :

1. Qu'est-ce que la mise à l'échelle implique dans les systèmes répartis ? **(1 pt)**
   Augmentation du nombre de processus
2. L'algorithme d'élection de Bully suppose-t-il qu'aucune panne de processus ne peut survenir pendant la procédure de l'élection tant que le nouveau coordinateur n'a pas encore sélectionné ?
   Non, solution distribuée
3. Donner une définition des événements concurrents dans les systèmes répartis en utilisant la relation «Happened before» ?
   Non ((e → é) et (é → e))
4. Comment organiser les processus du système réparti dans l'algorithme d'exclusion mutuelle de Suzuki-Kasami basé sur l'approche de passage de jeton ?
   Maillage total
5. Proposer un événement en concurrence avec l'horloge vectorielle (5, 3, 7) ?
   (3, 5, 4)
6. Quelle est la différence entre l'algorithme d'exclusion mutuelle de Ricart-Agrawala et l'algorithme de Lamport en termes d'optimisation ?
   Minimiser le nombre de messages
7. Dans un système réparti, comment appelle-t-on *GS* et *m* si l'expression suivante est vérifiée : $\forall\ m/\ Send\ (m) \in GS\ \Leftrightarrow Receive\ (m) \in GS$?
   GS : état global fortement cohérent m : message échangé
8. Comment les deux propriétés de sûreté et d'équité sont-elles garanties entre chaque deux sous-ensembles de processus selon l'algorithme d'exclusion mutuelle de Maekawa ?
   ∀ i, j / (Rsi ∩ Rsj ≠ ø) ∧ (\|Rsi\| = \|Rsj\|)
9. Selon l'algorithme d'élection de Franklin, à quel moment un processus peut-il être considéré comme gagnant ?
   Lorsqu'il reçoit un des messages qu'il a envoyés
10. Quelles sont les informations supplémentaires apportées par les horloges matricielles par rapport aux horloges vectorielles ?
    La connaissance du temps logique des autres processus

Exercice 1 (4 pts) : Les matrices ci-dessous désignent la date matricielle d'un état global d'un système réparti :

HP1 =

9	2	1	1
2	9	1	0
1	2	6	1
1	1	0	4

HP2 =

7	2	1	1
1	8	1	0
1	2	5	0
1	1	0	3

HP3 =

5	2	1	1
0	6	1	0
1	2	9	3
0	1	0	5

HP4 =

6	2	0	1
1	7	1	0
1	2	6	2
1	1	0	8

1. Quel est le nombre total des événements qui ont eu lieu dans le système avant cette date ? **(0.5 pt)**
   9+8+9+8 = 34
2. Quel est le nombre total de messages qui ont été échangés dans le système jusqu'à cette date ?
   (2+1+1)+(1+1)+(1+2+2)+(1+1) = 13
3. Jusqu'à cette date, combien d'événements locaux ont eu lieu ?
   1+1+1+3 = 6

4. Quelle est la chronologie d'envoi des messages de P1 et P2 ?
   P1: P2-P2-P4-P3 P2: P3-P1
5. Quel est l'ordre dans lequel les messages ont été reçus dans P3 et P4 ?
   P3 : P2-P1 P4 : P1-P3-P3
6. Est-ce que l'état global du système à cette date est cohérent ?
   Non, incohérent, message orphelin P2 vers P1
7. Par rapport à cette date, quelle est la date du dernier état global cohérent transitaire ?
   HP1[2,1] = 1 HP1[1,1] = 8 HP1[2,2] = 8
8. Quelle est la date du dernier état global fortement cohérent, par rapport à cette date ?
   HP1[2,1] = 1 HP1[1,1] = 8 HP1[2,2] = 8 HP3[3,4] = 2 HP3[3,3] = 8

Exercice 2 (6 pts) : Proposer un serveur TCP/JAVA (Thread) permettant de calculer $(A^B)^C$, tel que *A*, *B* et *C* sont trois nombres entiers choisis par le client :

```
import java.net.*;
import java.io.*; (0.25 pt)

class TCPServer {
   public static void main(String argv[]) throws Exception {
      ServerSocket welcomeSocket = new ServerSocket(8000);
      Thread t = new Thread(new Accepter_clients(welcomeSocket));
          t.start();
   }
}
class Accepter_clients implements Runnable {
   ServerSocket socketserver;
   double p=1;
   int i=1;
   public Accepter_clients(ServerSocket s){
          socketserver = s;
   }
   public void run() {
      try {
         Socket clientSocket = socketserver.accept();
         DataInputStream inFromClient = new DataInputStream(new BufferedInputStream(clientSocket()));
         DataOutputStream outToClient = new DataOutputStream(clientSocket.getOutputStream());
         int A = inFromClient.readInt();
         int B = inFromClient.readInt();
         int C = inFromClient.readInt();
         while (i<=B*C) {
            p=p*A;
            i=i+1;
         }
         outToClient.writeDouble(p);
         clientSocket.close();
      }
      catch (SocketTimeoutException e) { }
      catch(SocketException e) { }
      catch (IOException e) {}
   }
}
```