Université de Chlef Mars 2017
Département Informatique
Filière : 1 ère Année Master - Ingénierie des Logiciels

Examen de rattrapage

Module de Systèmes d'exploitation

Durée 1H30

Exercice 1 :

Quelles critiques peut-on faire à l'algorithme du banquier ?.

Réponse :

- *Coûteux* : L'algorithme est en effet très coûteux en temps d'exécution et en mémoire pour le système Puisqu'il faut maintenir plusieurs matrices, et déclencher à chaque demande de ressource, l'algorithme de vérification de l'état sain qui demande mxn^2 opération. (m est le nombre de types de ressources et n est le nombre de processus).
- *Théorique* : L'algorithme exige que chaque processus déclare à l'avance les ressources qu'il doit utiliser, en type et en nombre. Cette contrainte est difficile à réaliser dans la pratique.
- *Pessimiste* : L'algorithme peut retarder une demande de ressources dès qu'il y a risque d'interblocage (mais en réalité l'interblocage peut ne pas se produire).

(4.5 points)

Exercice 2 :

Décrivez les opérations dont on doit disposer au minimum pour faire une communication entre des processus par la technique de la "mémoire partagée".

Réponse :

Au minimum, on doit disposer des fonctions (instructions) suivantes :

- Une instruction exécutée par le premier processus voulant établir une communication, pour **réserver une zone** en mémoire qui servira de zone d'échange. L'instruction doit préciser la taille de la zone.
- Une instruction pour **déposer un message** dans la zone . Elle est utilisée par tout processus voulant envoyer ou diffuser un message. L'instruction doit préciser l'émetteur, le ou les destinataires du message, son contenu, ...etc
- Une instruction pour **prélever un message** par le processus destinataire .
- Une instruction pour fermer et **libérer la zone d'échange** après son utilisation. Elle est exécutée par le dernier processus l'ayant utilisée.

(4 points)

Exercice 3 :

L'état d'un système, à un instant donné, est représenté par les matrices suivantes de l'algorithme du Banquier :

Processus	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

- Le système est-il dans un état sain ? . Justifiez.

Réponse :
On applique la partie "Vérification de l'état sain" de l'algorithme du Banquier.

Matrice MAX

	A	B	C	D
P0	0	0	1	2
P1	1	7	5	0
P2	2	3	5	6
P3	0	6	5	2
P4	0	6	5	6

Matrice NEED

	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

Matrice ALLOCATION

	A	B	C	D
P0	0	0	1	2
P1	1	0	0	0
P2	1	3	5	4
P3	0	6	3	2
P4	0	0	1	4

Matrice AVAILABLE

A	B	C	D
1	5	2	0

Etape 1 : Initialisation

Work

A	B	C	D
1	5	2	0

Finish

P0	P1	P2	P3	P4
F	F	F	F	F

Etape 2 : indice=0

Work

A	B	C	D
1	5	3	2

Finish

P0	P1	P2	P3	P4
T	F	F	F	F

Etape 2 : indice=2

Work

A	B	C	D
2	8	8	6

Finish

P0	P1	P2	P3	P4
T	F	T	F	F

Etape 2 : indice=1

Work

A	B	C	D
3	8	8	6

Finish

P0	P1	P2	P3	P4
T	T	T	F	F

Etape 2 : indice=4

Work

A	B	C	D
3	14	11	8

Finish

P0	P1	P2	P3	P4
T	T	T	T	F

Etape 2 : indice=4

Work

A	B	C	D
3	14	12	12

Finish

P0	P1	P2	P3	P4
T	T	T	T	T

L'état du **système est donc sain**.

(2 points)

- Une requête (0, 4, 2, 0) arrive du processus P1. Peut-on l'accorder ? Justifiez.

Réponse :

Matrice MAX

	A	B	C	D
P0	0	0	1	2
P1	1	7	5	0
P2	2	3	5	6
P3	0	6	5	2
P4	0	6	5	6

Matrice NEED

	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

Matrice ALLOCATION

	A	B	C	D
P0	0	0	1	2
P1	1	0	0	0
P2	1	3	5	4
P3	0	6	3	2
P4	0	0	1	4

Matrice AVAILABLE

A	B	C	D
1	5	2	0

Matrice REQUEST1

A	B	C	D
0	4	2	0

Partie "Traitement de la Requête" de l'algorithme du Banquier.
Etape 1 : Request1 <= Available ? (oui)
Etape 2 : Request1 <= Need1 ? (oui)
Etape 3 : Sauvegarde de l'état du système. Modification des Matrices :

Available := Available - Request1

$Allocation_i := Allocation_i + Request_i$

$Need_i := Need_i - Request_i$

Matrice ALLOCATION

	A	B	C	D
P0	0	0	1	2
P1	1	4	2	0
P2	1	3	5	4
P3	0	6	3	2
P4	0	0	1	4

Matrice NEED

	A	B	C	D
P0	0	0	0	0
P1	0	3	3	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

Matrice AVAILABLE

A	B	C	D
1	1	0	0

Partie "Vérification de l'état sain" de l'algorithme du Banquier.

Etape 1 : Initialisation

Work

A	B	C	D
1	1	0	0

Finish

P0	P1	P2	P3	P4
F	F	F	F	F

Etape 2 : indice=0

Work

A	B	C	D
1	1	1	2

Finish

P0	P1	P2	P3	P4
T	F	F	F	F

Etape 2 : indice=2

Work

A	B	C	D
2	4	6	6

Finish

P0	P1	P2	P3	P4
T	F	T	F	F

Etape 2 : indice=1

Work

A	B	C	D
3	8	8	6

Finish

P0	P1	P2	P3	P4
T	T	T	F	F

Etape 2 : indice=3

Work

A	B	C	D
3	14	11	8

Finish

P0	P1	P2	P3	P4
T	T	T	T	F

Etape 2 : indice=4

Work

A	B	C	D
3	14	12	12

Finish

P0	P1	P2	P3	P4
T	T	T	T	T

L'état du système étant sain, on peut donc accorder la requête au processus P1.

(4 points)

Exercice 4 :

On considère le problème du Producteur/Consommateur avec :

- 1 producteur,
- 1 consommateur
- le buffer est limité et circulaire

Ecrire en Java une solution à ce problème en utilisant les sémaphores (Soignez la présentation de votre programme).

Solution :

Le candidat peut définir la classe sémaphore, ou utiliser directement le package de netbeans java.util.concurrent.Semaphore.

Code :

```
package java_prodcons_exam;

import java.util.LinkedList;
import java.util.Queue;
import java.util.concurrent.Semaphore;

/**
 * @Dr Mourad LOUKAM
 * Examen de rattrapage , M1-IL
 * March 2017
 */

public class ProdCons {
   private static int N = 10;  //La taille du buffer
   private static Producteur p = new Producteur();     // Le thread Producteur
   private static Consommateur c = new Consommateur(); // Le thread Consommateur
   private static Semaphore empty = new Semaphore(N);  // Sémaphore compteur , représentant les cases vides
   private static Semaphore full = new Semaphore(0);   // Sémaphore compteur , représentant les cases pleines
   private static Semaphore mutex = new Semaphore(1);  // Sémaphore permettant l'accès en mutex aux cases

   private static Queue<Integer> buffer = new LinkedList<Integer>();  //Buffer
   private static boolean Arreter = false;


   private static class Producteur extends Thread {

      private int val = 0;
      public void run() {

         while (!Arreter)
         {
            try {
                  int Element = Produire_Message();   //Produire un élément
                  empty.acquire();
                  mutex.acquire();

                  Deposer(Element); // Ajouter un élément au buffer
                  System.out.println("Le producteur dépose : " + Element );

                  mutex.release();
                  full.release();

            } catch (InterruptedException e) { }
         }
      }

      private void Deposer(int e) {
          buffer.add(e);
      }

      private int Produire_Message() {
         return ++val;                //Produire un message (un entier)
```

```
    }
  }


  private static class Consommateur extends Thread {

    public void run() {

      while (!Arreter)
      {
        try {
            full.acquire();
            mutex.acquire();

            int Element = Prelever();
            System.out.println("Le Consommateur prélève : " + Element );

            mutex.release();
            empty.release();

            //Consommer l'élément

        } catch (InterruptedException e) {}
      }
    }

    private int Prelever(){        //Prélever un message du buffer
      int e = buffer.remove();
      return(e);
    }

  }


  public static void main(String args[]) {

    p.start(); // Lancer le producteur
    c.start(); // Lancer le consommateur

    try {
      Thread.sleep(3000); // Exécution pendant 3 secondes
    } catch (InterruptedException ex) {}

    Arreter = true; // Forcer les threads à s'arrêter

  }

}
```


(*5.5 points)*