Université de Chlef Janvier 2017

Département Informatique
Filière : 1 ère Année Master - Ingénierie des Logiciels

# Examen semestriel

# Module de Systèmes d'exploitation

Durée : 1H30

***Exercice 1 :* (6 points)**

1. *Expliquez le principe de la communication par "tubes" en IPC. Donnez ses avantages et ses inconvénients.*

*Réponse :*
Un tube est un dispositif de communication qui permet une communication à sens unique entre deux processus. Les données écrites sur l'« extrémité d'écriture » du tube sont lues depuis l'« extrémité de lecture ». Les tubes sont des dispositifs séquentiels; les données sont toujours lues dans l'ordre où elles ont été écrites.
Avantage : Les processus liés par un tube sont synchronisés automatiquement par le système d'exploitation : il régule le flux de données entre les deux processus, en fonction de leur rapidité respective .
Inconvénient :

- Le transfert est à sens unique . Si on veut réaliser un transfert bidirectionnel entre deux processus, on doit créer 2 tubes différents.
- Les processus doivent avoir un lien de parenté (de type père fils).

(2 points)

2. *Qu'est ce que la "virtualisation" pour les systèmes d'exploitation ? . Donnez ses avantages et ses inconvénients.*

*Réponse :*
La virtualisation de systèmes d'exploitation est une technique consistant à faire fonctionner en même temps, sur un seul ordinateur, plusieurs systèmes d'exploitation comme s'ils fonctionnaient sur des ordinateurs distincts.
*Avantages :*

- Utiliser un autre système d'exploitation sans redémarrer son ordinateur, afin d'utiliser des programmes ne fonctionnant pas sur un système d'exploitation particulier.
- Exploiter des périphériques ne fonctionnant pas dans un système spécifique mais fonctionnant dans d'autres systèmes d'exploitation ;
- Tester un système d'exploitation en cours de développement sans compromettre la stabilité des autres systèmes en cours d'exécution.
- Tester des logiciels dans des environnements contrôlés, isolés et sécurisés ;

Inconvénient :

La virtualisation des serveurs exige des machines de capacité importante (processeur, mémoire, ...) pour pouvoir répartir ces ressources entre les différents serveurs virtuels.

La préparation de l'opération de virtualisation elle-même qui peut être compliquée et exige un certain savoir faire.

(2 points)

3. *Expliquez le principe de chacune des méthodes de traitement des interblocages.*

*Réponse :*
Il existe essentiellement trois méthodes pour traiter le problème de l'interblocage : *Prévention, Evitement, Détection-Guérison.*

- **Prévention** : On élimine complètement le risque d'interblocage en faisant en sorte que l'une des ses quatre conditions d'apparition ne soit pas vérifiée.
- **Evitement** : On évite le risque d'interblocage en veillant à que le système évolue uniquement entre états sains.
- **Détection-Guérison** : On attend que l'interblocage arrive, on le détecte puis on applique une méthode de guérison.

(2 points)

***Exercice 2 :* (6 points)**

*Soient les 3 processus suivants :*

| *Processus P1* | *Processus P2* | *Processus P3* |
|---|---|---|
| *Début*<br>*Cycle*<br>*Afficher("A")*<br>*Fin Cycle*<br>*Fin* | *Début*<br>*Cycle*<br>*Afficher("B")*<br>*Fin Cycle*<br>*Fin* | *Début*<br>*Cycle*<br>*Afficher("C")*<br>*Fin Cycle*<br>*Fin* |

Ecrire un programme Java qui implémente les 3 processus sous forme de threads, et impose que l'affichage soit toujours dans cet ordre : ABCABCABCABC...

Réponse :
L'étudiant peut implémenter la classe sémaphore ou importer le package directement.

---

```
package java_threads_afficheur;

/**
 * @Dr Mourad LOUKAM exam, January 2017
 */
import java.util.concurrent.Semaphore;

public class Java_Threads_Exam {

   static Semaphore S1 = new Semaphore(1);
   static Semaphore S2 = new Semaphore(0);
   static Semaphore S3 = new Semaphore(0);

   //Classe implémentant le thread A
   static class ThreadA extends Thread  {
      char Char_to_Repeat;
      // Constructeur
      ThreadA (char C) {
         Char_to_Repeat = C;
      }

      // On redéfinit la méthode run()
      public void run () {
         while(true)  {
```

```
            try {
                S1.acquire();
                System.out.println(Char_to_Repeat);
                S2.release();
                Thread.sleep(100);
            } catch (InterruptedException e) { return; }
        }
    }
}

//Classe implémentant le thread B
static class ThreadB extends Thread  {
    char Char_to_Repeat;
    // Constructeur
    ThreadB (char C) {
        Char_to_Repeat = C;
    }

    // On redéfinit la méthode run()
    public void run () {
        while(true)  {
            try {
                S2.acquire();
                System.out.println(Char_to_Repeat);
                S3.release();
                Thread.sleep(100);
            } catch (InterruptedException e) { return; }
        }
    }
}

//Classe implémentant le thread C
static class ThreadC extends Thread  {
    char Char_to_Repeat;
    // Constructeur
    ThreadC (char C) {
        Char_to_Repeat = C;
    }

    // On redéfinit la méthode run()
    public void run () {
        while(true)  {
            try {
                S3.acquire();
                System.out.println(Char_to_Repeat);
                S1.release();
                Thread.sleep(100);
            } catch (InterruptedException e) { return; }
        }
    }
}

public static void main(String[] args) {
 // On instancie les threads
    ThreadA ta1 = new ThreadA ('A');
    ThreadB ta2 = new ThreadB ('B');
    ThreadC ta3 = new ThreadC ('C');

    // On démarre les trois threads
    ta1.start();
    ta2.start();
    ta3.start();
}
```

}

---

(6 points)

***Exercice 2 : (8 points)***

On considère N processus **$P_i$** et un processus **Maître** , dont le schéma est donné ci-dessous :

| Processus $P_i$ | Processus Maître |
|---|---|
| Début | Début |
| PA ; | MA ; |
| PB ; | MB ; |
| Fin. | Fin. |

- Les N processus $P_i$ et le processus Maître s'exécutent en parallèle.
- Chaque processus Pi exécute la partie d'instructions PA et se bloque.
- Après avoir terminé la partie d'instructions MA, le processus Maître attend que tous les processus Pi aient terminé chacun sa partie PA ; il poursuit alors l'exécution de la partie MB.
- Une fois la partie MB terminée, le processus Maître libère tous les processus $P_i$ bloqués, qui peuvent alors continuer leur exécution.

Proposez un schéma de synchronisation des processus $P_i$ et Maître en utilisant des sémaphores.

*Réponse* :

Déclarations :
SMaitre : semaphore (init à 0); //sémaphore pour bloquer les processus Pi
SP : semaphore(init à 0); //sémaphore pour bloquer le processus Maitre
mutex : semaphore (init à 1); //sémaphore pour protéger la variable partagée i
i : entier (init à 0); //compte le nombre de processus Pi
N constante représentant le nombre de processus Pi.

| Processus $P_i$ | Processus Maître |
|---|---|
| Début<br><br>PA ;<br><br>Wait (mutex)<br>i:=i+1;<br>Si i=N alors<br>Signal((SP)<br>Finsi<br>Signal(mutex);<br><br>Wait(SMaitre);<br>PB ;<br>Signal(SMaitre)<br><br>Fin. | Début<br><br>MA ;<br>Wait(SP);<br><br>MB ;<br><br>Signal(SMAitre)<br>Fin. |

(8 points)