Réseaux. TP 1. C et Java pour le développement d'applications réparties

I. Langage C

I.1. Pointeurs et structures

Ecrire un programme qui demande un nom de machine (argument en ligne de commande) puis qui affiche la liste de ses adresses IP (en notation décimale pointée pour Ipv4).

Vous pourrez utiliser les structures et fonctions suivantes :

```
struct in_addr {
   unsigned long          /* Adresse IP codée en entier long (non signé) */
};

struct hostent {
   char   *h_name;              /* Nom officiel de l'hôte.   */
   char   **h_aliases;          /* Liste d'alias.            */
   int    h_addrtype;           /* Type d'adresse de l'hôte. */
   int    h_length;             /* Longueur de l'adresse.    */
   char   **h_addr_list;    /* Liste d'adresses. Pointeur sur pointeur sur l'adresse IP codée en unsigned long */
}
```

struct hostent * gethostbyname(const char * name); /* A partir d'une chaîne contenant un nom d'hôte (ou une adresse IP, IPv4 en notation décimale pointée ou IPv6 en notation RFC 1884), retourne un pointeur sur une structure de type hostent. Retourne NULL si erreur. */

char * inet_ntoa (struct in_addr in); /* Convertit une adresse IP contenue dans une structure de type in_addr (typiquement un entier long) en chaîne */

Méthode : récupérer le nom d'hôte et appeler gethostbyname(...) pour initialiser une structure de type hostent. Parcourir (boucle) son élément h_addr_list en convertissant les unsigned long (= struct in_addr) en chaînes.

I.2. Création et synchronisation de processus

Ecrire un programme en C qui créé deux processus fils (fork()), le premier affichant les nombres entiers allant de 1 à 50 et le second de 51 à 100. L'affichage des nombres doit bien sûr être totalement ordonné.

I.3. Création et synchronisation de processus

Ecrire un programme en C qui affiche tous les nombres entiers allant de 1 à 100, de la façon suivante : le programme principal lui-même affiche les nombres pairs pendant qu'un processus fils est chargé de l'affichage des nombres impairs. Pour que l'affichage soit complètement ordonné, les deux processus doivent donc se "renvoyer la balle".

II. Langage Java

II.1. Petit programme Java

Voici un squelette de programme Java qu'il vous est demandé de compléter et de commenter.

```
interface PersonneInterface{
    public void vieillir();
    public void clonerPersonne(Personne p);
}

class Personne implements PersonneInterface{
    private String nom ;
    private String prenom ;
    private int age ;

    public Personne (String nom, String prenom, int age){
        /* Compléter le code ici */
```

```
    }

    public void vieillir(){
        age++;
    }

    public String retournerNom(){
        return this.nom;
    }

    public String retournerPrenom(){
        return this.prenom;
    }

    public int retournerAge(){
        return this.age;
    }

    public void majNom(String nom){
        this.nom=nom;
    }

    public void majPrenom(String prenom){
        this.prenom=prenom;
    }

    public void majAge(int age){
        this.age=age;
    }

    public void clonerPersonne(Personne p){
        /* Compléter le code ici */
    }

    public void afficheInformations(){
        System.out.println("Je suis la personne "+nom+" "+prenom+" et d'age "+age+ " ans");
    }

}

class Etudiant extends Personne {

    int numEtudiant;
    String nomUniversite;

    public Etudiant (String nom, String prenom, int age, int numEtudiant, String nomUniversite){
        /* Compléter le code ici */
    }

    public void afficheInformations(){
        System.out.println("Je suis l'etudiant "+retournerNom()+" "+retournerPrenom()+ " d'age
"+retournerAge()+" ans identifie par "+numEtudiant+" et etudiant a "+nomUniversite);
    }
}

public class MonProgramme {

    public static void main(String args[]){
        try{
            Personne p = new Personne("Martin", "Pierre", 30);
            Etudiant e = new Etudiant("Dupond", "Jacques", 20, 12345, "La Garde");
            p.afficheInformations();
            e.afficheInformations();

            /*
            Personne p1 = new Personne("Durand", "Jean", 60);
            Personne p2 = new Personne("Fournier", "Paul", 50);
            p1=p2;
            p2.vieillir();
            p2.vieillir();
```

```
            p1.vieillir();
            p1.afficheInformations();
            p2.afficheInformations();
            */

            /*
            Personne p3=new Personne("Roche", "Nicolas", 35);
            p.clonerPersonne(p3);
            p3.vieillir();
            p.afficheInformations();
            p3.afficheInformations();
            */

        }
        catch (Exception e){
            e.printStackTrace() ;
        }
    }
}
```

a) Compléter le code du constructeur de la classe Personne, lequel doit permettre d'initialiser les 3 champs de chaque nouvel objet de type Personne.
b) Compléter le code de la méthode clonerPersonne(...), lequel doit affecter les valeurs de l'objet courant à celui passé en paramètre.
c) Compléter le constructeur de la classe Etudiant.
d) Mettre en commentaires la méthode **vieillir()** de la classe Personne ? Que se passe t-il ? Pourquoi ? Décommentez.
e) Mettre en commentaires la méthode **afficheInformations()** de la classe Etudiant. Que se passe t-il ? Pourquoi ? Décommentez.
f) Et si on met en commentaire la méthode **afficheInformations()** de la classe Personne ? Décommentez.
g) On décommente la partie **/* Personne p1 … */** Qu'est ce qui est affiché ? Expliquez.
h) Et si on décommente la partie **/* Personne p3 … */** ?

II.2. Threads Java

Soit le programme Java suivant :

```
class MonThread extends Thread {
    private String nom ;
    private int debut;
    private int max ;
    private MonThread monAutreThread;

    public MonThread(String nom, int debut, int max){
        this.nom=nom;
        this.debut=debut;
        this.max=max;
    }

    public void indicationAutreThread(MonThread monAutreThread){
        this.monAutreThread=monAutreThread;
    }

     public /* synchronized */ void run() {

        try {

            // monAutreThread.join();

            /* if (nom.equals("monThread2"))
                  monAutreThread.join(); */

            for (int i=debut;i<=max; i++){
                System.out.println("i= "+i);
            }

        }
        catch (Exception e){
```

```
            e.printStackTrace() ;
        }
    }
}

public class MesThreads {

    public static void main(String args[]){
        try{
            MonThread monThread1=new MonThread("monThread1", 0, 50) ;
            MonThread monThread2=new MonThread("monThread2", 51, 100);

            monThread1.indicationAutreThread(monThread2);
            monThread2.indicationAutreThread(monThread1);

            monThread1.start();
            monThread2.start();
        }
        catch (Exception e){
            e.printStackTrace() ;
        }
    }
}
```

a) Qu'est ce que fait le premier thread déclaré dans la fonction **main()** ? Et le deuxième ? Qu'est ce que fait le programme ?
b) On rajoute le mot-clé **synchronized** sur la méthode run() ? Est-ce que cela change quelque-chose ? Pourquoi ?
c) On décommente l'instruction **monAutreThread.join();** Que se passe t-il ? Pourquoi?
d) Et si on décommente à la place :

```
if (nom.equals("monThread2"))
    monAutreThread.join();
```

e) Comment ralentir l'affichage ?

II.3.Threads Java et ressources critiques

```
import java.lang.Math;
import java.util.*;

class MonThreadLecture extends Thread {
    private MonTableau monTableau ;

    public MonThreadLecture (MonTableau monTableau){
        this.monTableau=monTableau;
    }

    public void run() {
        monTableau.lectureTab();
    }
}

class MonThreadEcriture extends Thread {
    private MonTableau monTableau ;

    public MonThreadEcriture (MonTableau monTableau){
        this.monTableau=monTableau;
    }

    public void run() {
        monTableau.ecritureTab();

    }
}

class MonTableau{
    int max=10;
    int tableau []=new int[max];
```

```
    public MonTableau(int max){
        this.max=max;
        this.tableau =new int[max];
    }

    public void lectureTab(){
        try{
            for (int i=0;i<max; i++){
                System.out.println("lecture a l'indice "+i+" => "+tableau[i]);
            }
        }
        catch (Exception e){
            e.printStackTrace() ;
        }
    }
    public void ecritureTab(){
        try{
            for (int i=0;i<max; i++){
                int nb= (int)(Math.round((Math.random())*10) ) ;
                tableau[i]=nb;
                System.out.println("ecriture a l'indice "+i+" => "+tableau[i]);
            }
        }
        catch (Exception e){
            e.printStackTrace() ;
        }
    }

}

public class MesThreads {

    public static void main(String args[]){
        try{

            int max=10;
            while (true){
                Scanner keyboard = new Scanner(System.in);
                System.out.print ("Entrez un max  : ");
                String s=keyboard.nextLine();
                try{
                    max = Integer.parseInt(s);
                    break;
                }
                catch (Exception e){
                    System.out.println ("Valeur entiere SVP ");
                    continue;
                }
            }
            MonTableau monTableau=new MonTableau(max);

            MonThreadLecture monThreadLecture=new MonThreadLecture(monTableau) ;
            MonThreadEcriture monThreadEcriture=new MonThreadEcriture(monTableau);

            monThreadLecture.start();
            monThreadEcriture.start();
        }
        catch (Exception e){
            e.printStackTrace() ;
        }
    }
}
```

Voici deux threads on l'un vient lire dans un tableau rempli par l'autre. Quel(s) problème(s) constatez-vous ? Comment y remédier ?