Correction eval

Exercice 1 :

Soit le schéma d'exécution suivant :

- Le petit-fils PF1 exécute "xcalc"
- Le fils F1, après avoir créé le petit fils 1, exécute "xedit"
- Le fils F2 exécute "ls -al"
- Le père attend la fin de ses fils avant de terminer

Ecrivez le programme C qui réalise une telle exécution.

```
int main(){
   int pidF1, pidF2, pidPF1;

   pidF1 = fork();              // Création du fils F1

   // Code du fils F1
   if(pidF1 == 0){
      pidPF1 = fork();          // Création du petit fils PF1

      // Code du petit fils PF1
      if(pidPF1 == 0){
         execlp("xcalc", "xcalc", NULL);
      } else {
         execlp("xedit", "xedit", NULL);
         wait(NULL);            // Attendre la fin du petit fils PF1
      }
   // Code du père (après la création de F1)
   } else {
      pidF2 = fork();           // Création du fils F2

      // Code du fils F2
      if(pidF2 == 0){
         execlp("ls", "ls", "-al" NULL);
      } else {
         wait(NULL);            // Attendre la fin du fils 1
         wait(NULL);            // Attendre la fin du fils 2
         printf("Le processus père se termine\n");
      }
   }
}
```

Exercice 2 :

Soit le programme C suivant :

```
#include <stdio.h>
#include <signal.h>

int main() {
        int i, pid;
        setbuf(stdout, NULL);
        printf("i = %d ", i);
        pid = fork();
        if (pid == 0) {
                printf("Je suis le (1)\n");
                i += 10;
        }
        else {
                printf("Je suis le (2)\n");
                i += 1;
        };
        i *= 2;
        printf("i = %d ", i);
}
```

1. Remplacer les mentions (1) et (2) par les valeurs adéquates (Père ou Fils).
2. Quel est l'affichage produit par ce programme ?
3. En utilisant les signaux, modifier le code de façon à ce que l'affichage du fils se fasse toujours après celui du père.

1)

(1) doit être remplacé par "Fils".
(2) doit être remplacé par "Père".

2)

i = 0 Je suis le Père
i = 2 Je suis le Fils
i = 20

3)

Exercice 3 :

Soit l'ensemble de processus suivants, où les contraintes de précédence sont données par le graphe ci-dessous :

1. Donner une solution utilisant les sémaphores pour synchroniser ces processus de manière à respecter les contraintes de précédence.
   Il ne s'agit pas d'écrire du code C, mais de simplement donner un algorithme pour chaque processus, P1 à P6, avec les appels aux primitives P et V nécessaires à la synchronisation.
   Vous ne manquerez pas, bien sûr, de préciser combien de sémaphores vous sont nécessaires et quelles sont leurs valeurs d'initialisation.

2. Il existe une solution n'utilisant pas plus de trois sémaphores.
   Laquelle (si vous ne l'avez pas trouvée à la question précédente) ?

1)	2)
V(S) : Attends que j'incrémente P(S) : incrémente le sémaphore 1 : V(Sem1); V(Sem2); V(Sem3); 2 : P(Sem1); V(Sem5); 3 : P(Sem2); V(Sem4); 4 : P(Sem3); V(Sem7); 5 : P(Sem5); P(Sem6); V(Sem6); 6 : P(Sem6); P(Sem7);	V(S) : Attends que j'incrémente P(S) : incrémente le sémaphore 1 : V(Sem1); 2 : P(Sem1); V(Sem2); 3 : P(Sem1); V(Sem2); 4 : P(Sem1); V(Sem3); 5 : P(Sem2); V(Sem3); 6 : P(Sem3);

Exercice 4 :

On s'intéresse à une application Java client/serveur reposant sur des sockets en mode TCP. Cette application va gérer un compteur.

```
public class Compteur {
     private int val = 0;

     public Compteur() {
     }

     public int get() {
          return this.val;
     }

     public void set(int val) {
          this.val = val;
     }
}
```

Le rôle du serveur est d'attendre une connexion client, de mettre à jour le compteur en lui ajoutant la valeur envoyée par le client et de retourner au client la valeur actualisée du compteur.
Le serveur est à l'écoute sur toutes ses interfaces et sur le port 8000.
La version du serveur proposée est une version séquentielle (mono client).

```
public class TCPServer {

 public static void main(String[] args) {
  final int PORT = 8000;

  ServerSocket      ss = null;
  Socket            s;
  BufferedReader    reader = null;
  PrintWriter       writer = null;
  Compteur          c = new Compteur();
  int               n, val;

  try {
   ss = new ServerSocket(PORT, 10);

   s = ss.accept();
   reader = new BufferedReader(new InputStreamReader(s.getInputStream()));
   writer = new PrintWriter(s.getOutputStream(), true);

   while (c.get() < 1000) {
     n = Integer.parseInt(reader.readLine());
     val = c.get();
     c.set(val + n);
     writer.println(val + n);
     System.out.println("Compteur : " + (val + n));
   }
   s.close();
   System.out.println("Serveur arrêté");
  }
  catch (IOException e) {
     e.printStackTrace();
  }
 }
}
```

Le rôle du client est de demander à l'utilisateur de saisir une valeur (entier relatif). Tant que la valeur saisie est différente de 0, il l'envoie au serveur afin que ce dernier mette à jour la valeur du compteur, lit la réponse du serveur et l'affiche.
Quand la valeur saisie est 0, le client termine.

1. Ecrivez le code du client

2. Le serveur doit maintenant être capable de prendre en charge plusieurs clients simultanément. Les threads seront mis en œuvre en héritant de Thread.
   Donnez le code modifié du serveur.

Un problème de concurrence d'accès au compteur peut se poser dès lors qu'au moins 2 clients tentent de mettre à jour le compteur.

3. Ajoutez dans le code du **serveur** une attente de 10s permettant de mettre en évidence ce problème

4. Comment proposez-vous, dans le code du serveur, de régler ce problème de concurrence d'accès.

1)

```
public class TCPClient {
    public static void main(String[] args) {
        final String SERVER_ADDRESS = "localhost";
        final int PORT = 8000;

        try (
            Socket socket = new Socket(SERVER_ADDRESS, PORT);
             BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
             BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in))) {

            int value;
            String input;

            while (true) {
                System.out.print("Entrez une valeur (0 pour terminer) : ");
                input = userInput.readLine();
                value = Integer.parseInt(input);

                writer.println(value);
                if (value == 0) break;

                String response = reader.readLine();
                System.out.println("Valeur du compteur : " + response);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2)
Ajouter un **while (true) {}** dans le **try** du **TCPServer**

```
try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Serveur démarré...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                new ClientHandler(clientSocket).start();
            }

        }
```

3)

Ajouter un **Thread.sleep();** dans le **run()**

```
while ((n = Integer.parseInt(reader.readLine())) != 0) {
        synchronized (compteur) {
            int newVal = compteur.get() + n;
            Thread.sleep(10000);
            compteur.set(newVal);
            writer.println(newVal);
            System.out.println("Compteur mis à jour : " + newVal);
        }
    }
```

4)

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class TCPServer {
    private static final int PORT = 8000;
    private static Compteur compteur = new Compteur();
    private static Lock lock = new ReentrantLock();

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Serveur démarré...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                new ClientHandler(clientSocket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static class ClientHandler extends Thread {
        private Socket socket;

        public ClientHandler(Socket socket) {
            this.socket = socket;
        }

        public void run() {
            try (BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                 PrintWriter writer = new PrintWriter(socket.getOutputStream(), true)) {

                int n;
                while ((n = Integer.parseInt(reader.readLine())) != 0) {
                    lock.lock();
                    try {
                        int newVal = compteur.get() + n;
                        compteur.set(newVal);
                        writer.println(newVal);
                        System.out.println("Compteur mis à jour : " + newVal);
                    } finally {
                        lock.unlock();
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```