UFR SSI
M1 INFO - INF 2160u
CONCURRENCE
Session I - 2018-2019

Luc Courtrai

date : mardi 8 janvier 2019

durée : 2h

1 Question de cours ? (4 pts)

-1- Une variable globale (en dehors d'une fonction) d'un programme C, est-elle partagée par 2 threads POSIX. Pourquoi ?

-2- Sous Linux, un fichier ouvert (par l'appel système open) par un processus est-il toujours ouvert et accessible par son processus fils (fork). Pourquoi ?

-3- L'opération `P` sur un sémaphore est-elle toujours bloquante ? Pourquoi ?

-4- Comment protéger une variable dans un segment partagé, d'un accès concurrent de deux processus lourds issus d'un programme C sous LINUX ?

-5- Deux threads Java appelant la même méthode d'instance protégée par la clause `synchonized` sur deux objets différents sont-ils en exclusion mutuelle ? Pourquoi ?

-6- Pourquoi l'appel aux méthodes `wait` ou `notify` doit il être inclus dans un bloc `synchronized` ?

-7- Comment stopper un processus (pas le tuer) sans faire un ctrl Z sur le terminal

-8- Dans les streams d'objets en Java, comment fusionner les résultats suite à l'appel à `parallel` ? exemple `sum`=IntStream.range(0,10).parallel() ...

2 Prinln (4 pts)

```
class Th extends Thread {
  static int sval=0;
  int val;
  public Th(int v) {val =v;};
  public void run() {
   sval++;
   System.out.println(val+" " + sval);
   while(val -- >= 0 ) {
      System.out.println(val+" "+sval);
      if (val % 2 == 0) {// pair
        Th unT = new Th (val);
        unT.start();
        try {unT.join();} catch (InterruptedException e){}
      };
   }
 }
 public static void main(String args[]){
   (new Th(2)).start();
 }
}
```

Quel est le résultat de ce programme sur la sortie standard ?

3 Opération Z sur les Sémaphores (6 pts)

Voici une implantation (des sémaphores en JAVA en utilisant les moniteurs.

```
public class Semaphore {
  int             count = 0;
  private LinkedList<Thread> lockThreads;
  public Semaphore(int i) {
    count = i;
    lockThreads=new LinkedList<Thread>();
  }
  public void P() throws InterruptedException{
    synchronized (Thread.currentThread()){
      synchronized (this) {
         if (count != 0) {count--; return;}
         lockThreads.add(Thread.currentThread());
      }
      Thread.currentThread().wait();
    }
  }
  public synchronized void V() {
    if (lockThreads.size() != 0) {
      Thread first= lockThreads.remove();
      synchronized(first){
        first.notify();
      }
    } else
        count++;
  }
```

Modifier cette classe pour ajouter l'opération Z. L'appel à cette opération bloque le thread courant si le compteur est différent de (Z)éro et jusqu'à ce que la valeur de ce compteur passe à 0 par l'opération P. Tous les threads bloqués dans Z sont alors réveillés.

Attention les threads bloqués dans Z ne doivent pas être mélangés avec ceux bloqués dans P.

```
/*  operation Z
* bloque le thread  courant si le compteur est différent de 0
* et cela jusqu a ce que la valeur de ce compteur passe à nulle */
public void Z();
```

4 Initialisation d'un serveur (6 pts)

On désire mettre en place une application client serveur en C, C++ ou Java. Le serveur doit s'initialiser avant d'offrir ses services. Les clients, qui demandent le service, doivent être bloqués tant que le serveur n'a pas fini sa phase d'initialisation.

Vous devez utiliser pour la synchronisation les sémaphores et uniquement les deux opérations P et V . Vous devez compléter ces trois bouts de code (...) .

```
COTE SERVEUR
//Debut du serveur
...  // a compléter
//Fin de la partie initalisation
... // a compléter
COTE CLIENT
//Avant la demande au serveur
... // a compléter
```

On suppose que le serveur démarre en premier et qu'aucun client ne peut être lancé avant le serveur (le serveur a fini la partie "Début du serveur").

Seule la partie synchronisation nous intéresse. vous pouvez utiliser du pseudo code du type C++ :

```
shared int  cmp =0;     // variable partagées (connue par TOUS les programmes)
shared semaphoreMutex(1)  // sémaphore partagé initialise à 1
mutex.P();  // operation P sur le sémaphore  mutex
mutex.V();
```