Travaux Dirigés Java Temps Réel: TD4

Les synchronisations entre threads temps réel

Le but de ce td est comprendre le comportement des protocoles de synchronisation temps réel (PIP et PCE).

Exercice 1. L'exclusion mutuelle

Deux thread s'exécutent à la même priorité (19). Chaque thread exécute 2 ut normales puis 4ut dans une méthode get().

1. Dans un premier temps (programme SynchroRTT1.java), les 2 threads ont un accès non synchronisé à la méthode get(). Ce cas correspond au code avec commentaires. Vous devez observer qu'il n y a pas d'exclusion mutuelle, donc pas de protection des données manipulées par la méthode get().

2. Dans un second temps, (programme SynchroRTT2.java), la méthode get() doit maintenant être utilisée de manière exclusive. Vous utiliserez soit la synchronisation de bloc soit la synchronisation de méthode (préférence au premier cas). Vous devez observer l'exclusion mutuelle.

```
public class SynchroRTT {
private final static Object Bleue = new Object();
public static final int NB_IT = 177500;

public static void main(String[] args) throws InterruptedException {

while (!RConsole.isOpen()){
        RConsole.open();
}

RealtimeThread t1 = new RealtimeThread(
        new PriorityParameters(19),
        new PeriodicParameters(
                new RelativeTime(1000,0),
                new RelativeTime(50000,0),
                new RelativeTime(50000,0),
                new RelativeTime(50000,0), null, null)) {

        public void run() {

                for(int j=0 ; j < NB_IT ; j++);// une seconde
                RConsole.print("T1 : - 1 seconde \n");
                for(int j=0 ; j < NB_IT ; j++); //une seconde
                RConsole.print("T1 : - 2 secondes \n");

                //synchronized (Bleue) {
                        SynchroRTT.get();
                //}
        }
};

t1.setName("T1");
t2.setName("T2");
```

```
t1.start();
t2.start();
t2.join(50000);
t1.join(50000);
RConsole.close();
}
// synchronized static void get(){
static void get(){
for(int i=1 ; i < 5 ; i++){
for(int j=0 ; j < NB_IT ; j++);//une seconde
RConsole.print(Thread.currentThread().getName()+" - section critique seconde"+ i +"\n");
};
}
}
```
X

Exercice 2. Inversion de Priorité

Dans cet exercice vous allez mettre en évidence l'héritage de priorité qui est une propriété du protocole PIP.
Le code a mettre en place doit correspondre au tableau suivant :

Tâche	Priorité	Activation	Coût	Période	Echéance
Tau1	12	0	2000N+4000B	40000	40000
Tau2	16	4000	2000N+4000B	40000	40000
Tau3	14	6000	4000N	40000	40000

Exercice 3. Interblocage avec PIP (InterPIP)

Dans cet exercice vous allez mettre en évidence le problème d'interblocage. Problème existant avec le protocole PIP lors de l'utilisation de ressources imbriquées.
Le code a mettre en place doit correspondre au tableau suivant :

Tâche	Priorité	Activation	Coût	Période	Echéance
Tau1	12	0	1000N+(4000B+4000J+4000B) +1000N	40000	40000
Tau2	16	3000	1000N+(4000J+4000B+4000J)+1000N	40000	40000

Exemple d'utilisation des ressources imbriquées. Dans cet extrait de code le thread T3 (extrait de code):

1. execute une seconde en dehors de toute section critique (code non synchronisé, non protégé),
2. execute deux secondes dans la section bleue
3. entre dans la section jaune sans libérer la section bleue (les deux ressources sont imbriquées)
4. exécute 3 secondes dans la section jaune, l
5. ibère la section jaune,
6. exécute deux secondes dans la section bleue
7. libère la section bleue.
8. execute une seconde en dehors de toute section critique

```
for(int j=0 ; j < NB_IT ; j++);//une seconde
RConsole.print("T3 : - 1 seconde hsc\n");
RConsole.print("T3 : - demande ressource Bleue\n");

synchronized(Bleue){
RConsole.print("T3 : - entre ressource Bleue\n");
for(int j=0 ; j < NB_IT ; j++);//une se
synchronized(Bleue){

        RConsole.print("T3 : - entre ressource Bleue\n");
        for(int j=0 ; j < NB_IT ; j++);//une seconde
        RConsole.print("T3 : - "+(1)+" seconde bleue\n");
        for(int j=0 ; j < NB_IT ; j++);//une seconde
        RConsole.print("T3 : - "+(2)+" seconde bleue\n");

        synchronized(Jaune){

                RConsole.print("T3 : - entre ressource Jaune\n");
                for(int j=0 ; j < NB_IT ; j++);//une seconde
                RConsole.print("T3 : - "+(1)+" seconde jaune\n");
                for(int j=0 ; j < NB_IT ; j++);//une seconde
                RConsole.print("T3 : - "+(2)+" seconde jaune\n");
                for(int j=0 ; j < NB_IT ; j++);//une seconde
                RConsole.print("T3 : - "+(3)+" seconde jaune\n");
                RConsole.print("T3 : - quitte ressource jaune\n");
        }
        for(int j=0 ; j < NB_IT ; j++);//une seconde
        RConsole.print("T3 : - "+(3)+" seconde bleue\n");
        for(int j=0 ; j < NB_IT ; j++);//une seconde
        RConsole.print("T3 : - "+(4)+" seconde bleue\n");
        RConsole.print("T3 : - quitte ressource bleue\n");
}
for(int j=0 ; j < NB_IT ; j++);//une seconde
RConsole.print("T3 : - 1 seconde hsc\n");
```

Vous devez observer l'interblocage.

Exercice 4. Protocole PCE - suppression de l'interblocage

Dans cet exercice vous allez utiliser le protocole PCE pour supprimer le problème d'interblocage. Vous ne changez pas le code des Threads. Vous n'avez qu'à préciser que le protocole utilisé est PCE lors de la création des objets de synchronisations (bleu et jaune).

Pour initialiser les objets de synchronisation avec PCE voir l'extrait suivant :

```
PriorityCeilingEmulation pce1 = PriorityCeilingEmulation.instance(19);
MonitorControl.setMonitorControl(Bleue,pce1);

PriorityCeilingEmulation pce2 = PriorityCeilingEmulation.instance(19);
MonitorControl.setMonitorControl(Jaune,pce2);
```