Travaux Dirigés Java Temps Réel: TD2

Threads Temps Réel et modèles d'activation

Le but de ce TD est de mettre en oeuvre deux types d'entités ordonnançantes de RTSJ: RealtimeThread et AsyncEventHandler et de comprendre leur modèle d'activation et d'ordonnnancement.

Exercice 1. Création d'un threads Temps Réel

Vous allez créer un thread temps réel simple en utilisant la classe RealtimeThread (surcharge de run()). Vous affecterez une priorité de 15 au thread T1 (utilisation de la classe PriorityParameters).

```
public class OneRTT {
public static final int NB_IT = 176500;
public static long deb;
public static long start1;
public static long end1;
public static void main(String[] args) {

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

RealtimeThread t1 = new RealtimeThread(new PriorityParameters(15)) {

        public void run() {

        RConsole.println("Entre dans 1 !");
        start1=System.currentTimeMillis();
        RConsole.println("Delai de preemption : "+ Long.toString(start1-deb) );

        for(int k=0 ; k < NB_IT ; k++) ;
        for(int k=0 ; k < NB_IT ; k++);
        for(int k=0 ; k < NB_IT ; k++);

        end1=System.currentTimeMillis();

        RConsole.println("Debut 1 : " + Long.toString(start1-deb));
        RConsole.println("Duree 1 : " + Long.toString(end1-start1));
        RConsole.println("Fin de 1 *");
        }
};

deb=System.currentTimeMillis();
RConsole.println("Debut");

t1.start();
try {
        t1.join(7000);
}
catch (InterruptedException ie) {};

RConsole.close();
}
}
```

Exercice 2. Création d'un realtime thread périodique

Vous allez tracer l'exécution du thread T1 et vérifier que son activation est bien périodique et que la période est bien celle indiquée dans les paramètre d'activation en observant les dates de début d'exécution des instances.

```
public class OneRTTPeriodic {
public static final int NB_IT = 176500;
public static long deb;
public static long start1;
public static long end1;

public static void main(String[] args) {

while (!RConsole.isOpen()){
        RConsole.open();
}
RealtimeThread t1 = new RealtimeThread(new PriorityParameters(15),new PeriodicParameters(
        new RelativeTime(0,0), /* OFFSET */
        new RelativeTime(5000,0), /* Période */
        new RelativeTime(2000,0), /* Coût */
        new RelativeTime(5000,0), /* Echéance */
        null, null)) {

public void run() {
int i=0;
        do{
                RConsole.println("Entre dans 1 !!!!!!!");
                start1=System.currentTimeMillis();
                RConsole.println("Delai de preemption : "+ Long.toString(start1-(deb+i*5000)) );

                for(int k=0 ; k < NB_IT ; k++) ;
                for(int k=0 ; k < NB_IT ; k++) ;

                end1=System.currentTimeMillis();
                RConsole.println("Debut 1 : " + Long.toString(start1-deb));
                RConsole.println("Duree 1 : " + Long.toString(end1-start1));
                i++;
                RConsole.println("Fin de 1 *******");
        }while(waitForNextPeriod() && i<3);
}
};
deb=System.currentTimeMillis();
RConsole.println("Debut");

t1.start();

try {
        t1.join(20000);
}
catch (InterruptedException ie) {};
RConsole.close();
}
}
```

Exercice 3. Création de deux realtime thread périodiques

Dans ce exercice vous allez créer 2 threads périodiques de même période et de même coût mais dont l'activation initiale est différente. Les caractéristiques exactes sont données dans le tableau suivant. La tâche Tau2 est la plus prioritaire elle activée avec un offset de 1000 millisecondes. Vous devez observer la préemption de la tâches Tau1.

Tâche	Priorité	Activation	Coût	Période	Echéance
Tau1	15	immédiate	2000	5000	5000
Tau2	17	1000	2000	5000	5000

Exercice 4. Création d'un AsyncEventHander Entité Temps Réel Apériodique/Sporadique

Dans cet exercice, vous allez créer une tâche périodique et une tâche apériodique de priorité plus élevée. La tâche apériodique sera activée par la tâche périodique (au milieu de l'exécution de cette dernière). Vous allez observer la préemption de la tâche périodique par la tâche apériodique. L'activation de la tâche apériodique se fait via un appel a la méthode fire() sur un objet Event. Le ou les handlers associés (addHandler()) est ou sont activés. Les handlers seront exécutés en respect de leur priorité.

```
import lejos.nxt.comm.*;
import lejos.realtime.*;
import lejos.nxt.*;
public class ThrAperiod {
public static final int NB_IT = 177500;
public static long deb;
public static long deb2;
public static long start1;
public static long end1;
public static long start2;
public static long end2;
public static AsyncEventHandler aeh;
public static AsyncEvent ae;
public static void main(String[] args) {

while (!RConsole.isOpen()){
        RConsole.open();
}
aeh = new AsyncEventHandler(){
        public void handleAsyncEvent(){
                RConsole.println("Entre dans thread aperiodique !!!!!!!");
                start2=System.currentTimeMillis();
                RConsole.println("Delai de preemption : "+(start2-deb2) );
                RConsole.println("Thread aperiodique active à : "+(start2-deb) +" du debut. ");
                for(int k=0 ; k < NB_IT ; k++);
                end2=System.currentTimeMillis();
                RConsole.println("Duree aperiodique : " + Long.toString(end2-start2));
                RConsole.println("Fin du thread aperiodique *******"); }
};

aeh.setSchedulingParameters(new PriorityParameters(25));
ae= new AsyncEvent();
```

```
ae.addHandler(aeh);
RealtimeThread t1 = new RealtimeThread(new PriorityParameters(15),new PeriodicParameters(
                        new RelativeTime(0,0),
                        new RelativeTime(4000,0),
                        new RelativeTime(3000,0),
                        new RelativeTime(4000,0), null, null)) {
public void run() {
            int i=0;
            do{
                        RConsole.println("Entre dans 1 !!!!!!!");
                        start1=System.currentTimeMillis();
                        for(int k=0 ; k < NB_IT ; k++);
                        deb2=System.currentTimeMillis();
                        ae.fire();
                        for(int k=0 ; k < NB_IT ; k++);
                        end1=System.currentTimeMillis();
                        RConsole.println("Debut 1 : " + Long.toString(start1-deb));
                        RConsole.println("Duree 1 : " + Long.toString(end1-start1));
                        i++;
                        RConsole.println("Fin de 1 ********");
            }while(waitForNextPeriod() && i<3);
}
};
RConsole.println("Debut");
deb=System.currentTimeMillis();

t1.start();
try {
t1.join(10000);
} catch (InterruptedException ex) {};
RConsole.close();
}
}
```

Exercise 5. Pilotage d'un moteur et d'une roue

Cet exercise est une approche concrete des tâches périodiques Temps-réel.
Vous utiliserez un moteur et une roue. Imaginez que la rouer va piloter le déplacement d'un robot. Le mouvement sera périodique. Le robot va se déplacer de 2 mètres toutes les 10 secondes. Vous utiliserez un RealtimeThread pour cela.

Vous avez besoin la classe MotorPort.

- Le tachometre est un instrument qui va mesurer la rotation de la roue. La méthode *getTachoCount() retourne le nombre de degrés de rotation de la roue. Vous devez penser à initialiser le compteur avant chaque utilisation.*

```
MotorPort.A.resetTachoCount();
```

- La méthode *controlMotor()* est utilisée pour démarrer et stopper le moteur elle utilise deux parametres: POWER (0 - 100) and FUNCTION (1=Forward; 2=Backward; 3=Stop). Dans l'exemple suivant le moteur est démarré et fonctionne tant que la roue n'a pas atteint les 10 rotations.

```
MotorPort.A.controlMotor(100, 1);
while (MotorPort.A.getTachoCount() <= 3600) {}
MotorPort.A.controlMotor(0, 3);
```

- Question: Quel est le coût du thread ?
- Question: Que se passe t il lorsque que la puissance du moteur est réduite.

Exercise 6. Creation of a CostOverrunHander

Apply the preceeding technic to create a CostOverrunHandler and associate it to the MotorThread. The CostOverrun handler will stop the wheel when a costOverrun event occurs. To generate the costoverrun event slow down the wheel (or block it).

Exercice 6. Création d'un CostOverrunHander

Un CostOverrunHandler est un AsyncEventHanndler attaché à un RealtimeThread. Il sera appelé lors de l'occurence d'un dépassement de Coût.

Vous allez créer CostOverrunHandler associé au Thread périodique de pilotage du moteur. Le CostOverrunHandler va stopper le moteur. Pour activer le handler il suffira de ralentir le moteur (ou freiner les roues) pour que le dépassement de coût se produise.