M. Lamine Bougueroua

JAVA RT

TP – Java Temps-réel

Le TP se déroulera sur une machine virtuelle. L'ensemble des exercices du TP seront testés sur une Fedora-FC8-i386 dans la première partie.
Dans la deuxième partie, les exécutions se feront sur une machine dotée d'un noyau temps réel.

Configuration d'Eclipse :

L'utilisation de l'IDE Eclipse pour faire le développement des différentes classes nécessaires dans le TP est possible. Il suffit de configurer Eclipse :

- En ajoutant la libraire *foundation.jar* dans votre projet Java.
- En ajoutant un nouveau « External tools » qui aura comme application la tjvm (/opt/timesys/rtsj_ri_1.4/bin/tjvm) et comme argument -Xbootclasspath=/opt/timesys/ rtsj_ri_1.4/lib/foundation.jar -Djava.class.path=/«le chemin de la classe à exécuter» « le nom de la classe à exécuter »

MODELES DE TACHES

Dans cette partie, nous allons étudier la mise en œuvre de tache temps réel.
Dans un système temps réel, les priorités attribuées aux taches sont classées dans deux catégories : les priorités inférieures à 10 et les priorités supérieures à 10. En effet, un Thread normale possèdera une priorité variant entre 1 et 10 tandis qu'un Thread temps réel possèdera une priorité supérieure à 10.
Lien utile : ***http://www.rtsj.org***

Exercice 1 : Configuration de l'environnement

Positionnement des variables d'environnement Linux pour RI et JAVA :

1) Créer un répertoire de travail.
2) Modifiez le *path* en ajoutant les liens vers les répertoires *bin* de Java et RI :
   Exemple : /opt/timesys/rtsj_ri_1.4/bin/ et /opt/j2sdk1.4.2_18/bin/
3) Modifiez le *classpath* en ajoutant les liens vers les répertoires *lib* de Java et RI :
   Exemple : /opt/TimeSys/ rtsj_ri_1.4/lib/foundation.jar et /opt/ j2sdk1.4.2_18/lib/

   Créer un fichier de configuration RTenv.sh qui contient, en plus des deux lignes de commandes (2) et (3), un raccourci (alias) de la commande d'exécution suivante :

   tjvm -Xbootclasspath=/opt/timesys/ rtsj_ri_1.4/lib/foundation.jar **-Djava.class.path**= «le chemin de la classe à exécuter»

   Remarque : le chemin de la classe à exécuter sera le répertoire courant : **./**

Exercice 2 : temps d'exécution d'une unité de temps

Au cours de la mise en pratique, il nous faut calculer le temps d'exécution du Thread. Pour cela, on peut utiliser les classes qui gèrent le temps.

Ces classes se trouvent dans le package *javax.realtime*.

```
public void run(){
        AbsoluteTime start=new AbsoluteTime();
        AbsoluteTime end=new AbsoluteTime();
        Clock clock=Clock.getRealtimeClock();
        RelativeTime duree=new RelativeTime();

        start=clock.getTime(); //la valeur de retour est en milliseconde
                // boucle d'exécution vide
                for (int i=1;i<nombre_itérations;i++);
        end=clock.getTime();
        duree =end.subtract(start);
}
```

1) Déterminez approximativement le nombre d'itérations nécessaire afin de simuler l'exécution d'une unité de temps équivalente à une seconde.

Exercice 3 : Thread non temps-réel

Cet exercice consiste à vérifier le comportement des Threads non temps-réel selon les valeurs de leur priorités.

1) On commence par la création de deux Threads. On supposera que chaque Thread s'exécute pendant 10 unités de temps en affichant un message : « Salut je suis le Thread ***numéro***, j'en suis à ma ***nombre*** unités de temps».
2) Quelle est la priorité de chaque Thread?
3) Vérifiez l'ordre d'exécution des deux Threads lorsqu'ils ont des priorités différentes inférieures à 10.
4) Le deuxième Thread est un Thread temps réel (une priorité supérieure à 10), il sera lancé avec un retard de 2 unités de temps par rapport au premier Thread.
5) Les deux Threads sont temps réels (le premier moins prioritaire que le deuxième).

Exercice 4 : Thread Temps-Réel (Tâche périodique)

Les tâches périodiques sont implantées en attribuant un modèle d'arrivée périodique (*PeriodicParameters*) au paramètre *ReleaseParameters* d'une tâche temps-réel (*RealTimeThread*).

1) Créez un Thread non temps-réel qui s'exécute pendant 10 unités de temps en affichant un message : «Thread : j'en suis à ***numéro*** unités de temps».
2) Ajoutez une tâche périodique qui s'exécute après 3 secondes par rapport au Thread, et qui affiche un message à chaque 3 secondes : «Tache périodique : je passe avant toi ».

Exercice 5 : Thread TR (Tâche apériodique)

Pour mettre en œuvre une tâche apériodique, vous écrivez une classe SimpleAperiodic qui hérite d'un *AsyncEventHandler*. Le traitement que doit faire la tâche apériodique ne doit pas être décrit dans la méthode *run()* mais plutôt dans la méthode *handleAsyncEvent()*.
Le constructeur de la classe SimpleAperiodic doit avoir les paramètres suivant :

- Un modèle d'arrivé apériodique (*AperiodicParameters*) ;
- Une priorité d'exécution supérieure à 10 (*PriorityParameters*).

1) Créez un Thread non temps-réel qui s'exécute pendant 10 unités de temps en affichant un message : «Thread : j'en suis à ***numéro*** unités de temps».
2) Ajoutez une tâche apériodique qui s'exécute après quelques secondes par rapport au Thread, et qui affiche un message : «Tache apériodique: je passe avant toi ».
3) L'*AsyncEvent* joue le rôle de déclencheur de la tâche apériodique en utilisant sa méthode *fire*. Il faut noter qu'il est nécessaire d'associer l'objet SimpleAperiodic à l'objet *AsyncEvent* grâce à la méthode *addHandler :*
   AsyncEvent nom ;
   *nom**.addHandler (***SimpleAperiodic*)) ;*

- Faites quelques déclenchement de la tâche apériodique.

Exercice 6 : Thread TR (Tâche sporadique)

La mise en œuvre d'une tâche sporadique est identique à celle d'une tâche apériodique sauf que cette fois-ci le modèle d'arrivée est sporadique (*SporadicParameters*).

1) Créez une tâche sporadique qui s'exécute pendant 1 unité de temps, avec une période d'activation minimale égale à 2 unités de temps.
2) Déclenchez à plusieurs reprises la tâche sporadique, en utilisant la méthode *fire*.
3) Que se passe t'il lorsque :
   a. La durée entre deux déclenchements est supérieure à la période minimale.
   b. La durée entre deux déclenchements est inférieure à la période minimale.

Exercice 7 : Tâche sporadique – période minimale

On ne peut pas activer une tâche sporadique plus d'une fois pendant sa période minimale. Plusieurs solutions sont disponibles pour gérer ce genre de situation, une levée d'exception à titre d'exemple.
Cette solution est implantée par la modification du modèle d'arrivée sporadique :

```
…..
SporadicParameters SP =(SporadicParameters)sporadique_tache.getReleaseParameters();
SP.setMitViolationBehavior(SporadicParameters.mitViolationExcept);
sporadique_tache .setReleaseParameters(SP);
.....
```

1) Reprendre l'exercice 6 en utilisant le mécanisme d'exception pour détecter le cas de plusieurs activations dans la même période.

Exercice 8 : Timesys RT Linux & Jtime : Tâche sporadique

Reprendre l'exercice 6 sur la machine : **florence.**

1) Configurez les paramètres d'environnement :
- Transférez le fichier de configuration RTenv.sh sur florence :

sftp login@florence
put Rtenv.sh

- Connectez-vous sur florence :
  ssh login@florence
- Remplacez:
  /opt/timesys par /opt/timesys/jtime_1.0
  /opt/j2sdk1.4.2_18 par /usr/share/j2sdk1.4.2_02

2) Testez les 4 solutions suivantes :
   SP.setMitViolationBehavior(SP.**mitViolationExcept**);
   SP.setMitViolationBehavior(SP.**mitViolationIgnore**);
   SP.setMitViolationBehavior(SP.**mitViolationReplace**);
   SP.setMitViolationBehavior(SP.**mitViolationSave**);

Exercice 9 : étude de la faisabilité

Nous allons étudier la faisabilité d'un ensemble de tâches à travers l'exemple suivant :

Tâche	Durée d'exéc.	Echéance	Période
Thread 1	2	7	7
Thread 2	3	11	11
Thread 3	5	13	13

Toutes les valeurs sont exprimées en seconde.

1) Vérifier la faisabilité par la méthode addTo*Feasibility* de la classe *RealtimeThread.*
2) Développer une méthode de faisabilité basé sur la condition nécessaire et suffisante.

INDEX

Constructeurs d'un Thread temps réel :

```
RealtimeThread()
```

```
RealtimeThread(SchedulingParameters scheduling)
```

```
RealtimeThread(SchedulingParameters scheduling, ReleaseParameters release)
```

```
RealtimeThread(SchedulingParameters scheduling, ReleaseParameters release,
MemoryParameters memory, MemoryArea area, ProcessingGroupParameters group,
java.lang.Runnable logic)
```

Modèle d'arrivée des taches :

```
AperiodicParameters(RelativeTime cost,                RelativeTime deadline,
AsyncEventHandler overrunHandler, AsyncEventHandler missHandler)
```

```
PeriodicParameters(HighResolutionTime start,          RelativeTime period,
RelativeTime cost, RelativeTime deadline, AsyncEventHandler overrunHandler,
AsyncEventHandler missHandler)
```

SporadicParameters(RelativeTime interarrival, RelativeTime cost, RelativeTime deadline, AsyncEventHandler overrunHandler, AsyncEventHandler missHandler)

Constructeurs d'un AsyncEventHandler

AsyncEventHandler()
AsyncEventHandler(boolean nonheap)
AsyncEventHandler(boolean nonheap, java.lang.Runnable logic)
AsyncEventHandler(java.lang.Runnable logic)
AsyncEventHandler(SchedulingParameters scheduling, ReleaseParameters release, MemoryParameters memory, MemoryArea area, ProcessingGroupParameters group, boolean nonheap)
AsyncEventHandler(SchedulingParameters scheduling, ReleaseParameters release, MemoryParameters memory, MemoryArea area, ProcessingGroupParameters group, boolean nonheap, java.lang.Runnable logic)
AsyncEventHandler(SchedulingParameters scheduling, ReleaseParameters release, MemoryParameters memory, MemoryArea area, ProcessingGroupParameters group, java.lang.Runnable logic)