May 2009, spring semester.

TR52 Middle Exam
Principles and mechanisms for embedded real-time applications

Please, read with a **great attention** the instructions for each exercises and write the answers to each part in **two separate** sheets.

PART 1

1. VxWorks exercise (3 points)

Let us consider an intelligent vehicle system that tries to avoid obstacles on the road. Two sensors send data information to an embedded computer. The embedded VxWorks application sends the new speed and direction command to the intelligent vehicle. The computation task waits the data information from the two sensors, deduces a solution and sends it directly the vehicle. The two sensors have not the same refreshment time.

Explain your **VxWorks solution** to this problem. Try to find an architecture for exemple based on several RTP, task and communication.

2. Real Time Linux : questions (1 points)

Explain the preemption system used by the RTAI Linux and the difference between RT tasks and classical linux tasks.

3. Statecharts exercise (8 points)

Realize a **statechart** model of the following system.

Description :

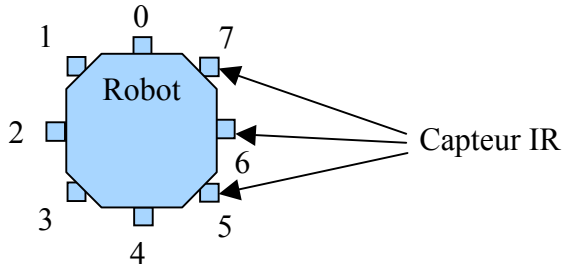
The best mobile phone in the world is the IPHONE. This mobile phone allows to make a phone call. This action preempts any running application.

You could run an Apple application like music, calendar, … or download another application. If there is a call during the execution of an apple application, when the call terminates, the system returns to the state before the call. On the contrary, with a download application the system return to a classical start state.

The battery level indicates the application priority. If the level is low (5%) the system is suspended and only the apple applications can run. If the battery level is under 2% only the call function can be run. Finally, if the level is under 1% the system is stopped until the level is above 5% and returns to the start state.

PART 2

Un robot est constitué de 8 capteurs infra-rouge pour détecter des obstacles proches. Ces capteurs sont positionnés sur le pourtour du robot qui est de forme circulaire (cf schéma).



Le robot explore son environnement en poursuivant une direction *d* correspondant au numéro d'un de ses 8 capteurs IR. Si un obstacle est détecté par un des capteurs *d*, *d-1* (mod 8) ou *d+1* (mod 8), le robot modifie sa trajectoire de la façon suivante :

- obstacle sur le capteur *d-1* (mod 8) : déplacement en direction de *d+1* (mod 8)
- obstacle sur le capteur *d+1* (mod 8) : déplacement en direction de *d-1* (mod 8)
- obstacle sur le capteur *d* : déplacement en direction de *d-1* (mod 8) ou de *d+1* (mod 8), en fonction du capteur ayant renvoyé la distance la plus éloignée
- obstacle sur les capteurs *d-1* (mod 8) et *d+1* (mod 8) : désactivation des capteurs IR, déplacement dans la direction *d+4* (mod 8) pendant une durée de 5s, réactivation des capteurs IR et finalement déplacement dans la direction *d+2* (mod 8).

Un tableau de nombres réels de dimension 8, nommé *tabIR,* est utilisé comme mémoire partagée pour stocker les informations lues par les capteurs. Une variable boolenne *doDetection* est également partagée pour que les threads de détection se mettent en attente si elle passe à la valeur *false* (mode capteurs IR désactivés).

D'une part, un thread principal, nommé *Explorer,* fait avancer le robot dans une direction *d*. Si un obstacle est détecté à moins de 10 cm, le robot change de direction en accord avec la stratégie d'exploration définie précédemment. Lorsqu'une direction est définie, le thread se met en attente de détection d'un autre obstacle à proximité. D'autre part, 8 threads de périodicité 600 ms sont associés à l'acquisition de l'état des capteurs IR, un thread par capteur. L'information lue sur le capteur numéro *n* (*n* compris entre 0 et 7) est stockée à l'indice *n* dans le tableau *TabIR*. Dans le cas où un retard se produit qui fait manquer la période suivante du thread, on décide de ne pas faire de lecture et d'attendre une période suivante, sans retard.

On suppose que l'on dispose des classes suivantes :

```
class Moteur {
        public on (int d) ; // démarre les moteurs dans la direction d
        public off () ; // stoppe les moteurs
        ...
}
class CapteurIR {
        public CapteurIR (int d) ; // construit un capteur associé à la direction d
        public getMeasure () ; // renvoie la mesure par le capteur (cm)
        ...
}
```

Ecrire les classes java temps réel correspondant à cette application.