TP IOT Programmation Contiki

A) Consultation fichier, Installation Contiki (cf docs)

B) Approche des programmes

Definition Les ProtoThreads (Wikipedia)

Les Protothreads fonctionnent comme des threads légers et sans pile, ou coroutines, fournissant un contexte de blocage à moindre coût en utilisant une mémoire minimale par protothread (de l'ordre d'un seul octet).

Un inconvénient est qu'on ne peut pas faire confiance aux variables locales dans le protothread pour avoir conservé leurs valeurs à travers un rendement vers un autre contexte. Ils doivent conserver leur état grâce à l'utilisation de variables statiques ou externes, souvent globales.

Un avantage est qu'ils sont très légers et donc utiles sur des systèmes à forte contrainte de mémoire comme les petits microcontrôleurs où d'autres solutions sont peu pratiques ou moins souhaitables.

Dans un programme executable pour Cooja en C, on retrouve les processus dans l'ordre suivant suivant :

-PROCESS(hello_world_process, "Hello world process");

Definition du processus.

-AUTOSTART_PROCESSES(&hello_world_process);

Lancement du processus defini. *Le nom prit en paramètre doit être identique de celui défini dans le processus précédent avec un « & » juste devant.*

-PROCESS_THREAD(hello_world_process, ev, data)

Semblable au Main en java. Il contient les processus Begin et End. *Le nom prit en paramètre doit être identique de celui défini dans le premier processus.*

-PROCESS_BEGIN();

Début du Process Thread.

-PROCESS_END();

Fin du Process Thred

C) Compilation

Nous pouvons compiler nos programmes directement depuis un terminal pour le bon fonctionnement des motes. Les commandes suivant se feront depuis le dossier. Le mot « hello-world » sera remplacé par le nom de vos programmes personnels

```
cd contiki/examples/hello-world
```

On peut tester le code dit « natif » pour vérifier le bon fonctionnement d'un programme et s'il n'utilise pas de réseau.

Ex : on compile en faisant make et on exécute hello-world.native, le terminal envoi « Hello World »
make hello-world vide créé make hello-world TARGET=native et d'autres fichiers.

```
make hello-world
./hello-world.native
```

Si on veut utiliser un réseau, alors on utilise minimal.net qui créera un hello-world.minimal-net
Une fois executé, le RPL sera activé mais le terminal nous enverra une erreur de droit d'accès.

```
make hello-world TARGET=minimal-net
./hello-world.minimal-net
```

Pour compiler un programme sur une mote TelosB, alors on utilise le SKY
Le fichier donner par la commande qui suit est de type binaire, on ne peut donc pas l'exécuter depuis un terminal mais uniquement depuis les motes.

```
make hello-world TARGET=sky
```

Le code obtenu peut être aussi utilisé sur les motes émulées depuis Cooja, ou transféré sur un vrai Telos en appelant :

```
make hello-world TARGET=sky hello-world.upload
```

On peut utiliser l'ipv6 en définissant la constante **UIP_CONF_IPV6=1**

Pour exectuer sur Cooja on réalise depuis le terminal. Cela lance directement une simulation sur cooja ! Incroyable non ?

```
make hello-world TARGET=cooja hello-world-examples.csc
```

Pour savoir ce que fait la simulaiton hello-world-examples.csc, nous vous invitons à regarder d'autres documents que nous avons rédigé sur le sujet.

D) Création des programmes

On me demande d'écrire un programme qui allume les 3 leds d'une mote. Je réalise donc le code suivant :

```
#include "contiki.h"
#include "leds.h" // For leds
#include <stdio.h> /* For printf() */
/*---------------------------------------
PROCESS(light_leds, "Allumer 3 leds");
AUTOSTART_PROCESSES(&light_leds);
/*---------------------------------------
PROCESS_THREAD(light_leds, ev, data)
{
  PROCESS_BEGIN();

        leds_on(LEDS_BLUE);
        leds_on(LEDS_RED);
        leds_on(LEDS_GREEN);

  PROCESS_END();
}
```

make leds.c TARGET=cooja

Les 3 leds s'allument.

On m'a ensuite demandé d'allumer les leds 1 par 1 avec un délai de 1 seconde.
Pour réaliser un délai, on réalise en premier un static struct avant le PROCESS_BEGIN() ;

```
static struct etimer et;
```

Une fois dans le begin, on set le timer avec etimer_set. On multiplie le temps par CLOCK_SECOND.
Attention, pour que le timer fonctionne sur Cooja, il ne faut pas oublier de mettre la speed à 100 %, de base la speed n'a pas de limite donc ignore le temps.

Le timer se découle avec le PROCESS_WAIT_EVENT_UNTIL. Avant de redécouler un timer, il faut penser à le remonter comme un réveil avec le etimer_set.

```
etimer_set(&et, 1 * CLOCK_SECOND);
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
```

le led_toggle('led') est une fonction qui allume la led si elle est etteinte et l'eteind si elle est allumée.
J'ai donc réaliser le code suivant :

```c
#include "contiki.h"
#include <stdio.h> /* For printf() */
#include "leds.h"
/*---------------------------------------------------------------------------*/
PROCESS(ledsAlter, "Allumer 3 leds");
AUTOSTART_PROCESSES(&ledsAlter);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(ledsAlter, ev, data)
{

  static struct etimer et; // Struct used for the timer
  static int nbtour;

  PROCESS_BEGIN();

        nbtour=0;
        while(1){

                leds_off(LEDS_BLUE)//eteind la led bleu (utile dans ce while)
                leds_on(LEDS_RED);//allume la led rouge

                etimer_set(&et, 1 * CLOCK_SECOND);//remonte le timer
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));//écoule le timer

                leds_off(LEDS_RED);
                leds_toggle(LEDS_GREEN);//allume la led verte car eteinte de base

                etimer_set(&et, 1 * CLOCK_SECOND);
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

                leds_toggle(LEDS_GREEN);//eteind la led verte car allumée juste avant
                leds_on(LEDS_BLUE);


                etimer_set(&et, 1 * CLOCK_SECOND);
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

                nbtour+=1;
                printf("Tour%d\n",nbtour);//amusement supp pour compter le nombre de tour d'allumage de leds.

        }



  PROCESS_END();
}
```