Programmer sur Contiki

Les Bases des Bases

Par Logan De Jesus

Définition des ProtoThreads

Les programmes sur les motes sont des threads.

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.

Structure d'un programme Contiki en C

```
PROCESS(hello_world_process, "Hello world process"); //Definition du processus.
AUTOSTART_PROCESSES(&hello_world_process); // Lancement du processus defini. Le nom en paramètre est l'adresse du processus défini auparavant, on indique donc un « & » avant le nom


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.
    //Votre Programme
    PROCESS_END(); //Fin du Process Thred
}
```

Compilation des Programmes Contiki

-Placer vous dans un dossier et placez le fichier le Makefile, Si vous ne le trouvez pas, voici son code :

```
CONTIKI_PROJECT = hello-world
all: $(CONTIKI_PROJECT)
#UIP_CONF_IPV6=1
CONTIKI = ../..
include $(CONTIKI)/Makefile.include
```

Note : #UIP_CONF_IPV6=1 permet d'activer l'ipv6 sur les motes

-Ensuite, créez votre nouveau programme en C.

-Pour compiler votre programme, ouvrez un terminal de commande et placez vous dans le dossier de votre nouveau programme (On rappelle qu'il doit contenir le makefile).

Compilation des Programmes Contiki

-Si votre code n'utilise pas de fonctionnalité de motes TelosB compilez en .native

```
make myprog TARGET=native
```

Vous pouvez l'éxecuter directement avec :

```
./myprog.native
```

-Si votre code utilise des fonctionnalités de motes TelosB compilez en .sky

```
make myprog TARGET=sky
```

-Si votre code utilise le réseau compilez en .minimal-net

```
make myprog TARGET=minimal-net
```

-Si la compilation ne renvoie pas d'erreurs, vous êtes alors prêts à tester votre programme sur des motes en utilisant le simulateur Cooja. Pour ce faire, on compile en .cooja

```
make myprog TARGET=cooja
```

Precessus Wait

Pour réaliser un délai, on déclare un static struct avant le PROCESS_BEGIN() ;

```
PROCESS_THREAD(ledsAlter, ev, data)
{

  static struct etimer et; // Struct utilisée pour le timer
```

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.

```
etimer_set(&et, 1 * CLOCK_SECOND);//remonte le timer
```

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 (étape d'avant)

```
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));//écoule le timer
```

Precessus Wait

Voici un programme qui print « bonjour » toutes les 5 secondes :

```c
#include "contiki.h"
#include <stdio.h> /* pour printf() */
#include "leds.h"
/*---------------------------------------------------------------------------*/
PROCESS(test, "Bonjour 5sec");
AUTOSTART_PROCESSES(&test);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(test, ev, data)
{

  static struct etimer et; // Struct utilisée pour le timer

  PROCESS_BEGIN();

    while(1){

        etimer_set(&et, 5 * CLOCK_SECOND);//remonte le timer
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));//écoule le timer

        printf("Bonjour\n");

    }

  PROCESS_END();
}
```

Utilisation des leds

Pour utiliser les leds, on importe la librairie leds.h

```
#include "leds.h"
```

Sur une mote, on retrouve 3 leds : Rouge, Verte, Bleu. Elle sont respectivement attachées aux noms suivant dans les programmes : LEDS_RED , LEDS_GREEN , LEDS_BLUE

On peut agir de plusieurs sur ces leds. On peut utiliser LEDS_ALL pour toutes les leds

Allumer :

```
leds_on(LEDS_RED);//allume la led rouge
```

Eteindre :

```
leds_off(LEDS_ALL);//eteint toutes les leds
```

Allumer/Eteindre : //Allume les leds si éteintes, les éteints si allumées

```
leds_toggle(LEDS_GREEN);
```

Utilisation des leds

Voici un programme qui allume les leds une par une toutes les secondes :

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

  static struct etimer et; // Struct utilisée pour 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();
}
```

Button des motes

Les sensors sont des capteurs.

En premier temps, le capteur de base est le button, nous avons besoins de la librairie dev/button-sensor.h `#include "dev/button-sensor.h"`

Après le début du procesus, on dit que nous allons utiliser les capteurs, ici le boutton avec :

```
SENSORS_ACTIVATE(button_sensor);//active les sensors
```

Il ne reste plus qu'à faire un déclancheur avec :

```
PROCESS_WAIT_EVENT_UNTIL(ev==sensors_event && data==&button_sensor);//Attendre jusqu'à que je clique
```

Le programme ne continuera pas tant que le button n'est pas pressé

Button des motes

Voici un programme qui allume les leds une par une quand on clique

```c
#include "contiki.h"
#include <stdio.h> /* pour printf() */
#include "leds.h"
#include "dev/button-sensor.h"
/*---------------------------------------------------------------------------*/
PROCESS(ledsAlter, "Allumer button leds");
AUTOSTART_PROCESSES(&ledsAlter);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(ledsAlter, ev, data)
{
  static int nbtour;
  PROCESS_BEGIN();

    nbtour=0;

    SENSORS_ACTIVATE(button_sensor);//active les sensors

    while(1){

        PROCESS_WAIT_EVENT_UNTIL(ev==sensors_event && data==&button_sensor);//Attendre jusqu'à que je clique

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

        PROCESS_WAIT_EVENT_UNTIL(ev==sensors_event && data==&button_sensor);

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

        PROCESS_WAIT_EVENT_UNTIL(ev==sensors_event && data==&button_sensor);

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

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

    }
  PROCESS_END();
}
```

Conclusion : Voici un programme qui Alterne les leds toutes les 4 secondes ou quand on clique sur le boutton !

```c
#include "contiki.h"
#include <stdio.h> /* pour printf() */
#include "leds.h"
#include "dev/button-sensor.h"
/*---------------------------------------------------------------------------*/
PROCESS(ledsAlt, "Allumer button leds");
AUTOSTART_PROCESSES(&ledsAlt);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(ledsAlt, ev, data)
{
  static int nbtour;
  static struct etimer et;

  PROCESS_BEGIN();

    nbtour=0;

    while(1){

        etimer_set(&et, 4 * CLOCK_SECOND);
        PROCESS_WAIT_EVENT_UNTIL((etimer_expired(&et)) || (ev==sensors_event && data==&button_sensor));


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

        etimer_set(&et, 4 * CLOCK_SECOND);
        PROCESS_WAIT_EVENT_UNTIL((etimer_expired(&et)) || (ev==sensors_event && data==&button_sensor));

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

        etimer_set(&et, 4 * CLOCK_SECOND);
        PROCESS_WAIT_EVENT_UNTIL((etimer_expired(&et)) || (ev==sensors_event && data==&button_sensor));

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

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

    }

  PROCESS_END();
}
```

Fin des programmes basiques