Rapport TP1 : IN412

Table des matières

Introduction

Le but de ce TP est la création de tâches RT ainsi que l'étude de leur exécution périodique.

I. Exercice 1 - « Hello World » en RTAI

fig 1. Code

```
/**
 *  Hello World RTAI
 */
#include <linux/module.h>
MODULE_LICENSE("GPL");
#include <asm/io.h>

#include <asm/rtai.h>
#include <rtai_sched.h>

#define NUMERO      1
#define PRIORITE    1
#define STACK_SIZE  2000
#define PERIODE     1000000000    //  1 s
#define TICK_PERIOD 1000000    //  1 ms
#define N_BOUCLE    10

static RT_TASK ma_tache;

/**
 *       Fonction du hello world.
 */
void attente(int arg) {

  RTIME t1,t2,dt;
  static int boucle = N_BOUCLE ;
  t1 = rt_get_time_ns();


  while (boucle--)
  {
        rt_task_wait_period();
        t2 = rt_get_time_ns();
        dt = t2-t1;

        printk("Durée de la boucle %d : %llu!\n",10-boucle,dt);
  }
```

```
}

/**
 *      Fonction générant la tache.
 */
static int mon_init(void) {

  int ierr;
  int iret;

  rt_set_oneshot_mode();

  //Initiaisation de la tache
  ierr = rt_task_init(&ma_tache, mon_code, NUMERO,
                      STACK_SIZE, PRIORITE, 0, 0);
  //Verification de la creation
  printk("[tache %d] cree code retour %d par programme %s\n",
         NUMERO,ierr,__FILE__);

  if (!ierr) {
    //Demarrage du compteur.
    start_rt_timer(nano2count(TICK_PERIOD));
    //Rendu de la tache périodique.
    iret = rt_task_make_periodic(&ma_tache, rt_get_time(), nano2count(PERIODE));

  }
  return ierr;
}

/**
 *      Fonction gérant à la sortie de la
 *      tâche la suppression de celle-ci ainsi
 *      que l'arrêt du timer.
 */
void mon_exit(void) {

  stop_rt_timer();
  rt_task_delete(&ma_tache);

}


module_init(mon_init);
module_exit(mon_exit);
```

fig 2. Trace console

```
<pc4105a 18:54 exo1> dmesg | tail
RTAI[sched]: Linux timer freq = 300 (Hz), CPU freq = 1866795000 hz.
RTAI[sched]: timer setup = 999 ns, resched latency = 2944 ns.
[tache 1] cree code retour 0 par programme
/home/rtai/Archive/exo/exo1/task.c
Durée de la boucle 1 : 999999344!
Durée de la boucle 2 : 1999999371!
Durée de la boucle 3 : 2999998959!
Durée de la boucle 4 : 3999998513!
Durée de la boucle 5 : 5000010498!
Durée de la boucle 6 : 5999998630!
```

On observe donc que la période de la tâche correspond environ bien à 1s. Au cours de cet exercice, nous avons utilisé la commande make pour compiler task.c à l'aide du makefile fournis ainsi que la commande ./run task s'occupant du chargement et déchargement du module.

II. Exercice 2 - « Génération de charge processeur »

Pour générer de la charge processeur, on écrit une boucle qui incrémente un compteur avec un nop à l'intérieur.

III. Exercice 3 - « Ordonnancement de tâches »

III. 1. Ordonnancabilité

On considère les jeux de taches suivants :

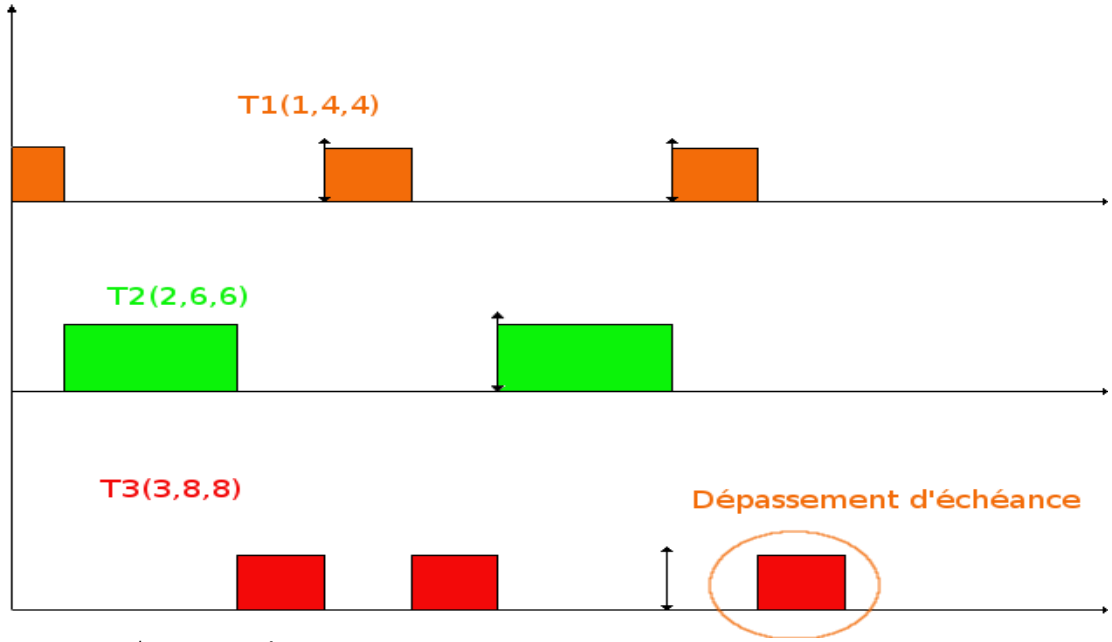
T1 :	C=1	Période = 4	Echéance = 4
T2 :	C=2	Période = 6	Echéance = 6
T3 :	C=3	Période = 8	Echéance = 8

T1 :	C=2	Période = 7	Echéance = 7
T2 :	C=2	Période = 11	Echéance = 11
T3 :	C=5	Période = 13	Echéance = 13

1er Jeu de tâches :

On calcule la borne d'ordonnançabilité. Ici ∑ci/pi = 23/24<1, on en déduit donc que l'ordonnancement est possible. On a ∏(ci/pi + 1) = 55/24 > 2, on ne peut donc pas statuer quand à l'ordonançabilité avec RM. On peut ordonnancer avec EDF.

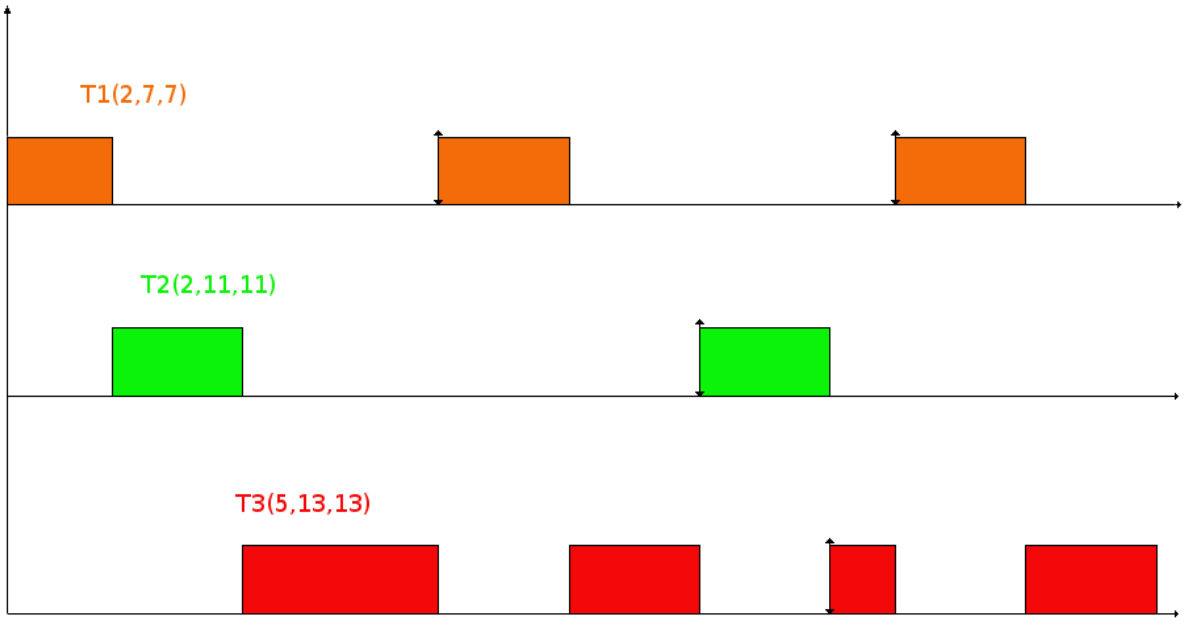
Fig 3. Ordonnancement du 1er ordonnancement de tâche



2nd Jeu de tâches :

On calcule la borne d'ordonnançabilité. Ici ∑ci/pi = 853/1001<1, on en déduit donc que l'ordonnancement est possible. On a ∏(ci/pi + 1) = 162/77 > 2, on ne peut donc pas statuer quand à l'ordonançabilité avec RM. On peut ordonnancer avec EDF.

Fig. 4 Ordonnancement en EDF du 2nd jeu de tâche



L'hyperpériode étant trop importante, on ne peut pas statuer sur l'ordonnançabilité du 2nd jeu de tâche.

III. 2. Mise en pratique

Il fallait donc implémenter l'ordonnancement des tâches en code C. Nous sommes donc partit avec notre implémentation proposé dans l'exercice 2, ce qui fût une perte de temps : l'opération nop implémenté en multitâche introduit des délais incohérent dans la fonction générant de la charge entrainant des délais de plus de 10 minutes. Une fois cette instruction supprimé, nous avons remarqué que la contrainte de temps n'était plus satisfaite. Nous avons donc passé une partie de notre temps à chercher la raison, ceci était lié au compilateur qui sans une opération de retour d'un opérande juge la fonction inutile et simplifie les boucles.

Fig. 5 code en RM

```
/*
 *    Ordonnancement de 3 tâches en priorités fixes.
 */

#include <linux/module.h>
MODULE_LICENSE("GPL");
#include <asm/io.h>

#include <asm/rtai.h>
#include <rtai_sched.h>

#define N_TASK 3
#define STACK_SIZE  2000
#define TICK_PERIOD 1000000    //  1 ms
#define BASIC_PERIOD 6500000
#define N_BOUCLE    4

static int C[N_TASK] = {1,2,3}, periode[N_TASK] = {4,6,8};
static RT_TASK task[N_TASK];
static RTIME timeUnit, origine;
static RTIME debut[N_TASK], fin[N_TASK];

/**
 *  Fonction générant de la charge
 */
int attente(int arg) {

  int decrement = BASIC_PERIOD, j = 0;

  while (arg--)
  {
```

```
        while(decrement--)
        {
                if (decrement%2) j++;
                else j+=2 ;
        }
  }
  return j ;
}

/**
 * Fonction de routine0
 */
void routine0(int i) {
   int j ;
    printk("Debut de la tache %d a %llu\n", i,rt_get_time()-origine);
    j = attente(2);
    printk("Fin de la tache %d a %llu\n", i, rt_get_time()-origine);
    rt_task_wait_period(); //ordonnancement a priorite fixe
    // a remplacer par rt_task_set_resume_end_times(-periode,-dealine); pour un
ordonnancement a priorite variable a algorithme EDF
  // ici rt_task_set_resume_end_times(-C*timeUnit, -C*timeUnit);
}

/**
 * Fonction de routine1
 */
void routine1(int i) {
    int j
    printk("Debut de la tache %d a %llu\n", i,rt_get_time()-origine);
    j = attente(2);
    printk("Fin de la tache %d a %llu\n", i, rt_get_time()-origine);
    rt_task_wait_period(); //ordonnancement a priorite fixe
    // a remplacer par rt_task_set_resume_end_times(-periode,-dealine); pour un
ordonnancement a priorite variable a algorithme EDF
  // ici rt_task_set_resume_end_times(-C*timeUnit, -C*timeUnit);
}

/**
 * Fonction de routine2
 */
void routine2(int i) {
    int j ;
    printk("Debut de la tache %d a %llu\n", i,rt_get_time()-origine);
    j = attente(2);
    printk("Fin de la tache %d a %llu\n", i, rt_get_time()-origine);
    rt_task_wait_period(); //ordonnancement a priorite fixe
    // a remplacer par rt_task_set_resume_end_times(-periode,-dealine); pour un
ordonnancement a priorite variable a algorithme EDF
```

```
  // ici rt_task_set_resume_end_times(-C*timeUnit, -C*timeUnit);
}
/**
 * Fonction fournissant la mesure de temps.
 */
void etalonnage (void) {
  //Etalonnage de l'unité de temps d'une tâche
  int j ;
  RTIME t1,t2;
  t1 = rt_get_time();
  j = attente(1);
  t2 = rt_get_time();
  timeUnit = t2-t1;
  printk("L'unite de temps de cette simulation vaut : %llu\n",timeUnit);
}

/**
 *  Fonction initialisant les différentes tâches.
 */
static int mon_init(void) {

  int i;

  int ierr[N_TASK], iret[N_TASK];
 static void *  routine[N_TASK]  = {routine0, routine1, routine2} ;
  rt_set_oneshot_mode();

  for (i=0; i<N_TASK; i++){
    ierr[i] = rt_task_init_cpuid(&(task[i]), routine[i], i, STACK_SIZE, i+1, 0, 0, 0);
    printk("[tache %d] cree code retour %d par programme %s\n",
           i,ierr[i],__FILE__);
  }
    if (!ierr[0]&&!ierr[1]&&!ierr[2]) {
        start_rt_timer(nano2count(TICK_PERIOD));
        etalonnage();
        origine = rt_get_time();
      for (i=0; i<N_TASK; i++){
        iret[i] = rt_task_make_periodic(&task[i], origine, periode[i]*timeUnit);
      }
    }
  return ierr[0]&&ierr[1]&&ierr[2];
}




/**
```

```
 *  Fonction de sortie des tâches.
 */
void mon_exit(void) {
  int i;
  stop_rt_timer();
  for (i=0; i<N_TASK; i++){
    rt_task_delete(&task[i]);
  }
}


module_init(mon_init);
module_exit(mon_exit);
```

Pour obtenir le code en EDF, on décommente juste les lignes préparé dans le code ci-dessus.

```
[tache 0] cree code retour 0 par programme /home/rtai/Desktop/TP1/task.c
[tache 1] cree code retour 0 par programme /home/rtai/Desktop/TP1/task.c
[tache 2] cree code retour 0 par programme /home/rtai/Desktop/TP1/task.c
L'unite de temps de cette simulation vaut :
Debut de la tache 0 a 2186
Fin de la tache 0 a 27332746
Debut de la tache 1 a 27336264
Fin de la tache 1 a 81992881
Debut de la tache 2 a 81994891
Debut de la tache 0 a 109567128
Fin de la tache 0 a 136895727
Debut de la tache 1 a 164350565
Fin de la tache 1 a 219007265
Debut de la tache 0 a 219134005
Fin de la tache 0 a 246462601
Fin de la tache 2 a 273311037
Debut de la tache 2 a 273311855
Debut de la tache 0 a 328700950
Fin de la tache 0 a 356029512
Debut de la tache 1 a 356030884
Fin de la tache 1 a 410687374
Fin de la tache 2 a 437288614
Debut de la tache 2 a 438269162
Debut de la tache 1 a 493051354
Fin de la tache 1 a 547707986
Fin de la tache 2 a 574921582
Debut de la tache 2 a 657405134
Fin de la tache 2 a 739435134
```

On pratique de même avec le second jeu de tâche.

Conclusion

Nous avons pût voir au cours de ce TP comment ordonnancer différentes tâches au sein d'un OS temps réel (Linux RTAI) et adapter les algorithmes appris en cours.