Rapport TP2 : IN412

Table des matières

Introduction

Le but de ce TP est la synchro-communication de tâches RT et la communication avec les processus utilisateur.

I. Exercice 1 - « Tâche avec sémaphore »

Fig. 1 Code du premier exercice

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

#include <asm/rtai.h>
#include <rtai_sched.h>
#include <rtai_sem.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 task1,task2;
static SEM sem1, sem2;


void code1(int arg){

        int ierr1,ierr2,ierr3;
        //rt_change_prio(rt_whoami(), rt_get_prio(rt_whoami())+1);
        rt_printk("T1\n");
        ierr1 = rt_sem_signal(&sem2);
        ierr2 = rt_sem_wait(&sem1);
        rt_printk("T1 apres T2\n");
        ierr3 = rt_sem_signal(&sem2);
}

void code2(int arg){

        int ierr1,ierr2,ierr3;
        rt_change_prio(rt_whoami(), rt_get_prio(rt_whoami())+1);
```

```
        ierr1 = rt_sem_wait(&sem2);
        rt_printk("Demarre apres T1\n");
        ierr2 = rt_sem_signal(&sem1);
        ierr3 = rt_sem_wait(&sem2);
        rt_printk("T2 finie\n");
}

static int mon_init(void) {
  int ierr1,ierr2;
  rt_set_oneshot_mode();
  rt_typed_sem_init(&sem1, 0 ,BIN_SEM );
  rt_typed_sem_init(&sem2, 0 ,BIN_SEM );

  ierr1 = rt_task_init(&task1, code1, NUMERO,
                       STACK_SIZE, PRIORITE, 0, 0);
  printk("[tache %d] cree code retour %d par programme %s\n",
         NUMERO,ierr1,__FILE__);

  ierr2 = rt_task_init(&task2, code2, NUMERO,
                       STACK_SIZE, PRIORITE, 0, 0);
  printk("[tache %d] cree code retour %d par programme %s\n",
         NUMERO,ierr2,__FILE__);

  if (!ierr1 && !ierr2) {

    start_rt_timer(nano2count(TICK_PERIOD));
    rt_task_resume(&task1);
    rt_task_resume(&task2);

  }
  return ierr1&ierr2;
}

void mon_exit(void) {

  stop_rt_timer();

  rt_sem_delete(&sem1);
  rt_sem_delete(&sem2);

  rt_task_delete(&task1);
  rt_task_delete(&task2);

}


module_init(mon_init);
```

```
module_exit(mon_exit);
```

On observe bien qu'en modifiant les priorité, l'odre d'exécution des tâches ne change pas grâce aux sémaphores.

Fig. 2 Trace console après exécution

```
T1
Demarre apres T1
T1 apres T2
T2 finie
```

II. Exercice 2 - « Watchdog »

Fig. 3 Code

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

#include <asm/rtai.h>
#include <rtai_sched.h>
#include <rtai_sem.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 task1,task2;
static SEM sem;
static int cmpt = 0;


void fonctionIncrement(int arg){
        int i = 0;
        RTIME tp;
        while(i < 7){
                tp = TICK_PERIOD*(100+(10*cmpt));
                rt_task_set_resume_end_times(-nano2count(tp),-nano2count(tp));
```

```
                cmpt++;
                rt_printk("[INFO] FonctionIncrement de %d - Compteur = %d, Periode =
%d\n",i, cmpt, tp);
                rt_sem_signal(&sem);
                i++;
        }
}

void watchdog(int arg){
        RTIME t,temps;
        int i = 0;
        while(i < 7){
                t = rt_get_time_ns();
                rt_sem_wait_timed( &sem, nano2count(150*TICK_PERIOD) );
                temps = rt_get_time_ns() - t;

                if(temps >= 150*TICK_PERIOD){
                        rt_printk("[ERREUR] Watchdog de %d - Task timed out!, Temp:
%llu\n",i, temps);
                }

                else{
                        rt_printk("[INFO] Watchdog de %d - Ok! - Temp: %llu\n",i, temps);
                }

                i++;
        }
}



static int mon_init(void) {
  int ierr1,ierr2;
  rt_set_oneshot_mode();
  rt_typed_sem_init(&sem, 0 ,BIN_SEM );

  ierr1 = rt_task_init(&task1, fonctionIncrement, NUMERO,
                       STACK_SIZE, 0, 0, 0);
  printk("[tache %d] cree code retour %d par programme %s\n",
         NUMERO,ierr1,__FILE__);

  ierr2 = rt_task_init(&task2, watchdog, NUMERO,
                       STACK_SIZE, PRIORITE, 0, 0);
  printk("[tache %d] cree code retour %d par programme %s\n",
         NUMERO,ierr2,__FILE__);
```

```
  if (!ierr1 && !ierr2) {

    start_rt_timer(nano2count(TICK_PERIOD));
    rt_task_make_periodic(&task1,rt_get_time(),nano2count(100*TICK_PERIOD));
    rt_task_resume(&task2);

  }
  return ierr1&ierr2;
}

void mon_exit(void) {

  stop_rt_timer();

  rt_sem_delete(&sem);

  rt_task_delete(&task1);
  rt_task_delete(&task2);

}

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

Fig. 4 Trace d' exécution console

```
[INFO] FonctionIncrement de 0 - Compteur = 1, Periode = 100000000
[INFO] Watchdog de 0 - Ok! - Temp: 100010865
[INFO] FonctionIncrement de 1 - Compteur = 2, Periode = 110000000
[INFO] Watchdog de 1 - Ok! - Temp: 109992208
[INFO] FonctionIncrement de 2 - Compteur = 3, Periode = 120000000
[INFO] Watchdog de 2 - Ok! - Temp: 119995290
[INFO] FonctionIncrement de 3 - Compteur = 4, Periode = 130000000
[INFO] Watchdog de 3 - Ok! - Temp: 130000825
[INFO] FonctionIncrement de 4 - Compteur = 5, Periode = 140000000
[INFO] Watchdog de 4 - Ok! - Temp: 139994957
[INFO] FonctionIncrement de 5 - Compteur = 6, Periode = 150000000
[INFO] Watchdog de 5 - Ok! - Temp: 149997434
[ERREUR] Watchdog de 6 - Task timed out!, Temp: 150002326
[INFO] FonctionIncrement de 6 - Compteur = 7, Periode = 160000000
```

Conclusion

Nous avons pû implémenter au cours de ce TP certains prototype de système utilisé dans des systèmes temps réel tel que les watchdogs. D'autre part nous avons pût revoir l'utilité des sémaphores dans l'éxécution de tâches en parallèle.