Examen d'IN412

Rémy Kocik, Nicolas Pernet

3 Novembre 2009

——— SANS DOCUMENTS ———

A LIRE AVANT DE COMMENCER :

- les tâches notées T_i sont des tâches périodiques et les tâches notées J_i sont des tâches apériodiques ;
- lorsque les dates d'activation ou phases ne sont pas mentionnées, c'est qu'il y a activation synchrone ;
- afin de dessiner les ordonnancements, des trames vous ont été remises. Pensez à indiquer ce que chaque axe représente (tâche T_n, exercice 11 ...), ainsi que votre nom sur chacune des feuilles ;
- ne construisez un ordonnancement que lorsque cela vous est demandé ! Tout recours à un ordonnancement (graphique) pour prouver quoi que ce soit (ordonnançabilité par exemple) sera noté comme faux ;

1 Exercice 1 : Vrai ou Faux

Répondez par vrai ou faux et justifiez :

- Tâches périodiques – Si un jeu de tâches périodiques Ψ à activation synchrone avec échéances égales aux périodes est ordonnançable avec Earliest Deadline First, il est ordonnançable avec Rate Monotonic. Vrai ou Faux ?
- Tâches périodiques – Soit un jeu de tâches périodiques Ψ à activation synchrone avec échéances égales aux périodes. Si $U_\Psi = 1$, Ψ est ordonnançable avec Earliest Deadline First. Vrai ou Faux ?
- Tâches apériodiques – La borne d'ordonnançabilité du serveur ajournable (deferrable server) est inférieure (plus restrictive) que celle du serveur à scrutation (polling server). Vrai ou Faux ?
- Tâches avec contraintes de ressources – Le priority inheritance protocol (PIP) permet d'éviter les interblocages (deadlocks). Vrai ou faux ?
- Gestion des surcharges – Un overrun cause obligatoirement un overload. Vrai ou Faux ?

2 Exercice : Cas de tâches avec contraintes de précédences

Question 1 - Représentez le graphe de précédence correspondant aux contraintes suivantes : $J_1 \rightarrow J_3$, $J_2 \rightarrow J_3$, $J_2 \rightarrow J_4$, $J_3 \rightarrow J_5$, $J_3 \rightarrow J_6$, $J_4 \rightarrow J_6$ et $J_4 \rightarrow J_7$.
Question 2 - Le tableau ci-dessous donne les caractéristiques des tâches. Déterminez les nouvelles valeurs des dates de réveil (r_i^*) et des échéances (d_i^*) afin de les ordonnancer avec Earliest Deadline First (EDF).

	J_1	J_2	J_3	J_4	J_5	J_6	J_7
C_i	2	4	3	5	2	2	5
r_i	0	1	6	4	6	8	12
d_i	12	11	16	16	15	25	19

Question 3 - Construisez l'ordonnancement EDF obtenu avec les nouvelles dates de réveil et les nouvelles échéances. Les contraintes sont-elles satisfaites ?

3 Exercice 3 : Tâches périodiques

Soit le jeu de tâches Ψ suivant à activation synchrone :

	T_1	T_2	T_3	T_4
C_i	1	3	1	3
D_i	5	17	4	8
P_i	5	24	4	10

Question 1 - Calculez l'hyperpériode de ce jeu de tâche.
Question 2 - Quel est l'algorithme optimal pour ce type de jeu de tâches si on se restreint aux algorithmes à priorités fixes ?
Question 3 - Statuez sur l'ordonnançabilité de ce jeu de tâches avec l'algorithme de la question 2 en calculant le pire temps de réponse de chaque tâche.
Question 4 - Quel est l'algorithme optimal pour ce type de jeu de tâches ?
Question 5 - Représentez sur l'intervalle [0, 24] l'ordonnancement obtenu en appliquant Earliset Deadline First à ce jeu de tâche. Cela vous suffit-il pour statuer sur l'ordonnançabilité du jeu de tâche avec Earliset Deadline First ?

4 Exercice 4 : Tâches périodiques

Soit le jeu de tâches suivant :

	T_1	T_2	T_3
C_i	1	5	5
$P_i = D_i$	3	16	24

Question 1 - Quelle est l'hyperpériode de ce jeu de tâche ?
Question 2 - En utilisant seulement les bornes d'ordonnançabilité sur le facteur d'utilisation, peut-on statuer sur l'ordonnançabilité de ce jeu de tâche avec Rate Monotonic ? Justifiez.
Question 3 - Ce jeu de tâches est-il ordonnançable avec Earliest Deadline First (EDF) ? Justifiez.
Question 4 - On veut rajouter une tâche T_4 de période P_4 telle que $P_4 = 50$. Quelle condition doit remplir la durée d'exécution C_4 de cette nouvelle tâche pour que le jeu de tâche reste ordonnançable avec EDF ? Quelle est alors la valeur entière maximale que peut prendre C_4 ?

5 Exercice 5

Soit le jeu de tâches périodiques suivant :

	T_1	T_2
C_i	1	1
$P_i = D_i$	4	5

Ce jeu de tâche est ordonnancé avec Rate Monotonic. On y ajoute un serveur, lui aussi ordonnancé selon Rate Monotonic, afin de pouvoir traiter des tâches apériodiques. Le serveur est de type Deferrable Server (serveur ajournable) a une capacité $C_s = 2$ et une période de $P_s = 6$, les tâches apériodiques y sont traités en FIFO. Surviennent alors les tâches apériodiques suivantes (rappel : r_i correspond à la date d'activation) :

	J_1	J_2	J_3	J_4
r_i	1	4	10	11
C_i	2	1	2	1

Question 1 - Représentez sur l'intervalle [0, 25] l'ordonnancement obtenu.
Question 2 - Donnez les temps de réponse constatés pour chacune des tâches apériodiques.
Question 3 - Citez deux autres techniques d'exécution de tâches apériodiques avec respect des contraintes de tâches périodiques.

6 PARTIE RTAI

On considère le code source du module RTAI fourni en annexe "annexe1.c". Ce module implante l'ordonnancement de 3 tâches temps réel.

Question 1 - Analyser le code de ce module, et à partir de cette analyse tracer l'ordonnancement obtenu à l'exécution dans l'intervalle de temps [now,now+40ms].

Question 2 - Quelles sont les modifications qui doivent être apportées à ce code pour obtenir un ordonnancement EDF à activation synchrone (avec Di=Pi)? Expliquez.

Question 3 - On ajoute maintenant dans le code de la tâche 1 et de la tâche 3 des accès à un sémaphore binaire. En prenant en compte le nouveau code des tâches 1 et 3 proposé ici, tracer l'ordonnancement obtenu à l'exécution dans l'intervalle de temps [now,now+40ms].

```
void mon_code1( int arg) {
  int ierr;
 while (1)
  {

    charge(2000000); //2ms
    ierr=rt_sem_signal(&ma_ressource);
    charge(1000000); //1ms
    ierr=rt_sem_signal(&ma_ressource);
    rt_task_wait_period();

  }
}

void mon_code3( int arg) {
  int ierr;
 while (1)
  {
    ierr=rt_sem_wait(&ma_ressource);
    charge(1000000); //1ms
     ierr=rt_sem_wait(&ma_ressource);
    charge(1000000); //1ms
    rt_task_wait_period();

  }
}
```

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

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



#define NUMERO1       1
#define NUMERO2       2
#define NUMERO3       3
#define PRIORITE1     3
#define PRIORITE2     4
#define PRIORITE3     2
#define STACK_SIZE  2000
#define PERIODE1     11000000     //  11 ms
#define PERIODE2     10000000     //  10 ms
#define PERIODE3      6000000     //  6 ms

#define TICK_PERIOD 1000000    //   1ms


static RT_TASK ma_tache1,ma_tache2,ma_tache3;
SEM ma_ressource;

void charge(RTIME val_charge)
{
  //on suppose que cette fonction permet d'attendre dans la tâche
  // un temps constant quelque soit la préemption et quelque soit la machine uti
lisée
  // (génération d'une charge constante)
  // val_charge doit être donné en ns
}

void mon_code1( int arg) {
  int ierr;
 while (1)
  {
    charge(2000000); //2ms
    rt_task_wait_period();

  }
}


void mon_code2( int arg) {
  int ierr;
 while (1)
  {
    charge(1000000); //1ms
    rt_task_wait_period();

  }
}



void mon_code3( int arg) {

 while (1)
```



```
  {
    charge(3000000); //3ms
    rt_task_wait_period();

  }
}



int init(void) {

  //Initialisation paramètres

  int ierr;
  RTIME now;



  rt_set_oneshot_mode();

  ierr = rt_task_init(&ma_tache1, mon_code1, NUMERO1, STACK_SIZE, PRIORITE1, 0,
0);
  ierr = rt_task_init(&ma_tache2, mon_code2, NUMERO2, STACK_SIZE, PRIORITE2, 0,
0);
  ierr = rt_task_init(&ma_tache3, mon_code3, NUMERO3, STACK_SIZE, PRIORITE3, 0,
0);

  rt_typed_sem_init(&ma_ressource, 0, BIN_SEM);

  if (!ierr) {

    start_rt_timer(nano2count(TICK_PERIOD));

    now = rt_get_time()+nano2count(1000000);
    rt_task_make_periodic(&ma_tache1, now,nano2count(PERIODE1));
    rt_task_make_periodic(&ma_tache2, now,nano2count(PERIODE2));
    rt_task_make_periodic(&ma_tache3, now,nano2count(PERIODE3));
  }
  return ierr;
}

void cleanup(void) {

  stop_rt_timer();

  rt_task_delete(&ma_tache1);
  rt_task_delete(&ma_tache2);
  rt_task_delete(&ma_tache3);
 rt_sem_delete(&ma_ressource);


}

MODULE_INIT(init);
MODULE_EXIT(cleanup);
```