Informatique sûre et temps réel
Examen – 2h00

Master SPI – M2 II & M2 I4

3 décembre 2019

Exercice 1 : Ordonnancement (/3)

Dire si les jeux de tâches suivants sont ordonnançables avec les algorithmes RMS et EDF :

1. $J_1 = \{\tau_1(3,7,7); \tau_2(2,11,11); \tau_3(6,13,13)\}$
2. $J_2 = \{\tau_1(1,3,3); \tau_2(5,18,18); \tau_3(4,27,27)\}$
3. $J_3 = \{\tau_1(4,9,9); \tau_2(5,18,18); \tau_3(5,27,27)\}$
4. $J_4 = \{\tau_1(3,9,9); \tau_2(8,18,18); \tau_3(5,27,27)\}$

Remarque : une tâche $\tau_i(C_i, D_i, T_i)$ est définie par sa charge maximale C_i, son échéance D_i et sa période T_i. On suppose que les dates d'activation sont identiques.

Exercice 2 : Partage de ressources (/4)

On considère un jeu de trois tâches à priorités statiques. Elles se partagent deux ressources protégées par des sémaphores d'exclusion mutuelle (sémaphores bleu et rouge). La séquence d'exécution pour chaque tâche est donnée dans le tableau ci-dessous.

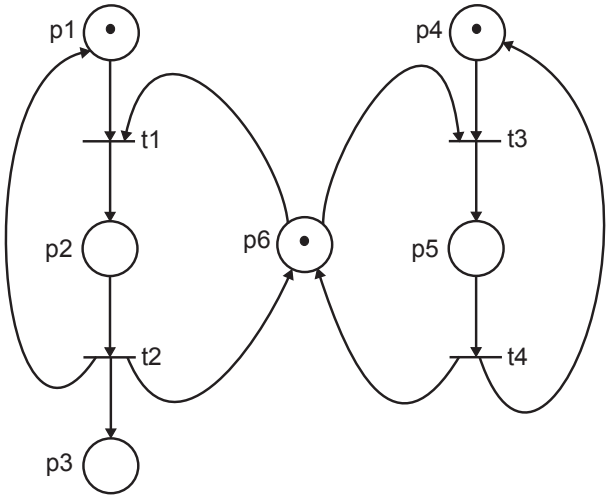
Q1 : Tracer le chronogramme d'exécution de $t_0 = 0$ à $t_f = 34$ des 3 tâches en utilisant l'algorithme RMS à priorités fixes. Quel(s) problème(s) identifiez-vous sur la séquence d'exécution ? (Dans cette question, il n'est pas demandé d'utiliser l'un des 3 algorithmes vus en cours : PIP, PCE ou PCP)

Q2 : On choisit d'utiliser l'algorithme PCE pour y remédier. Tracer le nouveau chronogramme d'exécution des 3 tâches. Les problèmes sont-ils tous résolus ?

tâche	date d'activation	séquence de la charge	échéance	période	priorité
τ_1	4	1+3(rouge)+4(bleu)+1	30	30	3
τ_2	2	1+3(rouge)+9	30	30	2
τ_3	0	1+6(bleu)+1	30	30	1

Exercice 3 : Analyse d'un réseau de Petri (/4)

On se propose d'analyser le réseau de Petri ci-dessous. Répondre aux questions suivantes :
— Le réseau est-il vivant ?
— Calculer les invariants (P-invariants et T-invariants).
— Le réseau est-il borné ?



Exercice 4 : Lampe pour les dyslexiques (/9)

En se basant sur les résultats de deux chercheurs [1], la start-up Lexilife vient de sortir une lampe à éclairage LED dont la fréquence de pulsation et la largeur d'impulsion sont réglables. Le réglage de ces deux paramètres permet de faciliter la lecture pour certains dyslexiques.

On se propose ici d'analyser plusieurs aspects du système à développer pour réaliser une version connectée de cette lampe en utilisant un système d'exploitation temps réel.

Ordonnancement des tâches (/3)

Le programme exécuté sur le calculateur de la lampe est composé de 3 tâches réalisant la fonction de réglage des paramètres via une application smartphone :
— Tâche 1 : la tâche de génération du signal PWM est périodique et fonctionne à une fréquence F réglable avec une valeur minimale de 20 Hz. Cette tâche utilise 1 ms maximum de charge de calcul sur le calculateur.
— Tâche 2 : la tâche de traitement des données issues du smartphone est cadencée à 10 Hz avec une charge maximale de 20 ms.
— Tâche 3 : la tâche de communication avec le smartphone est apériodique, elle est déclenchée seulement sur requête de l'utilisateur au niveau du smartphone. Sa charge maximale est de 600 ms.

On souhaite ordonnancer ce jeu de tâches avec l'algorithme RMS et avec la tâche apériodique exécutée en tâche de fond de faible priorité.

Q1 : Dans un premier temps, on ne considère que les tâches 1 et 2 en ignorant la tâche 3. Déterminer la plus haute fréquence F de la tâche 1 pour laquelle le jeu de deux tâches est théoriquement ordonnançable.

1. Le Floch, A., & Ropars, G. (2017). Left-right asymmetry of the Maxwell spot centroids in adults without and with dyslexia. Proceedings of the Royal Society B : Biological Sciences, 284(1865), 20171380.

Q2 : En supposant que la tâche 3 est de plus faible priorité que les deux autres tâches, on souhaite malgré tout que le délai d'exécution de cette tâche ne dépasse pas 1 s, ce qui signifie que le délai entre la requête et la fin d'exécution de la charge doit rester inférieur à la seconde. Déterminer la plus haute fréquence F possible dorénavant.

Exécution des tâches temps réel (/3)

Le code Xenomai fourni en annexe permet de simuler la génération du signal PWM avec les paramètres envoyés par l'application du smartphone. Pour cela, les trois tâches décrites ci-dessous sont créées.

- La tâche `pwm` (tâche 1) génère le signal PWM au niveau d'un port parallèle à partir des paramètres de réglage (fréquence et rapport cyclique) qui lui sont transmis.
- La tâche `traitement` (tâche 2) récupère les paramètres issus du smartphone, vérifie leur validité, et envoi les données à la tâche `pwm` à l'aide de boites aux lettres.
- Une application smartphone permet à l'utilisateur de faire une requête pour modifier les paramètres de réglage du signal PWM (la fréquence et le rapport cyclique). La tâche `comSmartphone` (tâche 3) permet de gérer la communication avec le smartphone. Pour cela, la fonction `wait_Smartphone` récupère les données envoyées par le smartphone. Cette fonction, dont l'implémentation n'est pas fournie ici, est bloquante, ce qui signifie que la tâche est bloquée tant que l'utilisateur ne fait pas de requête.

Q3 : Donner directement sur votre copie (et non sur l'annexe) le code qui doit être complété à l'endroit indiqué dans la tâche `pwm` pour transformer la fréquence reçue en période.

Q4 : Donner directement sur votre copie le code qui doit être complété à l'endroit indiqué dans la tâche `traitement` de façon à envoyer les données de fréquence et de rapport cyclique. Pour cela on utilisera les boîtes aux lettres `rtmbx_rcy` et `rtmbx_freq` (utiliser la fonction `rt_queue_write` vue en TP).

Q5 : Quelles sont les valeurs initiales de la fréquence et du rapport cyclique au lancement du programme ?

Production de la lampe (/3)

On souhaite modéliser par un réseau de Petri les différentes phases de fabrication de la lampe. Deux lignes de production parallèles sont utilisées pour la fabrication, elles réalisent les mêmes tâches et se partagent certains outils. Une ligne de production consiste en la réalisation de trois étapes de fabrication :

1. la première étape consiste à assembler plusieurs éléments de structure à l'aide d'un premier outil disponible en deux exemplaires (un par ligne de production).
2. Pour la deuxième étape, il s'agit d'ajouter la carte électronique et les câbles d'alimentation dans la structure. Cette tâche n'utilise pas d'outils particulier.
3. La troisième étape consiste à faire les tests de bon fonctionnement. Pour cela, un logiciel de test sur ordinateur est utilisé, il est disponible en un seul exemplaire à se partager.

Un seul opérateur s'occupe d'effectuer successivement toutes les étapes de sa ligne de production. Après avoir terminé la fabrication d'une lampe, il en fabrique une nouvelle.

Q6 : Faire le réseau de Petri de ce système. Marquer votre réseau de façon à ce que chaque opérateur soit à la première étape de fabrication.

```c
/*  Examen lampe    */
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <termios.h>
#include <sys/io.h>
#include <sys/mman.h>                   // for MCL_CURRENT and MCL_FUTURE
#include <stdbool.h>
#include <fcntl.h>

#include <rtdk.h>                       // for rt_printf, etc.
#include <native/task.h>
#include <native/timer.h>
#include <native/sem.h>
#include <native/mutex.h>
#include <native/queue.h>

// For the application
#define HIGHEST_PRIO    99              // 0 is the lowest, 99 the highest
#define PERIOD          1000000         // 1 ms
#define LPT             0x3050
int pinValue = 0xFF;                    // parport value (0 or 1)

// boites aux lettres
static RT_QUEUE rtmbx_freq;
static RT_QUEUE rtmbx_rcy;
// semaphores
static RT_MUTEX mut_smart;
// données issues du smartphone
int rcy_smart = 50;
double frequency_smart = 100;

// Tâche PWM
static void pwm(void *arg)
{
    int dutyperiod;
    int rcymsg = rcy_smart;
    double freqmsg = frequency_smart;
    int retval = 0;
    int myduty = 50;
    RTIME  now;
    RTIME period = 10*PERIOD;

    // Passage en tache periodique
    rt_task_set_periodic(
        NULL,           /* the current task */
        TM_NOW,         /* delay before release, TM_NOW = none */
        period          /* period in clock ticks */  );
    // Boucle principale
    while (true){
        // recupère le temps courant
        now = rt_timer_read();

        // ecrire sur le port parallèle
        outb(pinValue,LPT);

        // lit la boite aux lettre de fréquence
```



```c
        retval = rt_queue_read(&rtmbx_freq, &freqmsg, sizeof(freqmsg), TM_NONBLOCK);
        if(retval == sizeof(freqmsg)) //tous les octets recus = message présent
        {
            /* Q3 : déterminer la nouvelle periode */
            rt_task_set_periodic(NULL, TM_NOW, period);
        }

        // lit la boite aux lettre de rapport cyclique
        retval = rt_queue_read(&rtmbx_rcy, &rcymsg, sizeof(rcymsg), TM_NONBLOCK);
        if(retval==sizeof(rcymsg)) //tous les octets recus = message présent
        {
            myduty = rcymsg;
        }

        // calcul de la periode haute
        dutyperiod = (int)(myduty/100.0*period);
        // endormissement jusqu'a now + periode haute
        rt_task_sleep_until(now + dutyperiod);
        // inverse les données
        pinValue = ~pinValue;
        // ecrire sur le port parallèle
        outb(pinValue,LPT);
        // inverse les données
        pinValue = ~pinValue;

        rt_task_wait_period(NULL);
    }

    rt_printf("Fin PWM\n");
}

// Tâche TRAITEMENT
static void traitement(void *arg)
{
    int rcy = 0;
    double freq = 0;
    RTIME period = 100*PERIOD;

    // Passage en tache periodique
    rt_task_set_periodic(
        NULL,           /* the current task */
        TM_NOW,         /* delay before release, TM_NOW = none */
        period          /* period in clock ticks */  );
    // Boucle principale
    while ( true )
    {
        rt_mutex_acquire(&mut_smart, TM_INFINITE);
            if(rcy_smart > 0 && rcy_smart < 100) rcy = rcy_smart;
            if(frequency_smart > 20 && frequency_smart < 150) freq = frequency_smart;
        rt_mutex_release(&mut_smart);

        /* Q4 : envoi des messages */

        rt_task_wait_period(NULL);
    }

    rt_printf("Fin TRAITEMENT\n");
}
```

```c
// Tâche COMSMARTPHONE
static void comSmartphone(void *arg)
{
    int rcy = 50;
    double freq = 50;

    // Boucle principale
    while ( true )
    {
        // attente requête smartphone (bloquant)
        wait_Smartphone(&freq,&rcy);

        rt_mutex_acquire(&mut_smart, TM_INFINITE);
            rcy_smart = rcy;
            frequency_smart = freq;
        rt_mutex_release(&mut_smart);
    }

    rt_printf("Fin COMSMARTPHONE\n");
}

// Tâche MAIN
int main(void)
{
    RT_TASK task_main, task_traitement, task_comSmartphone, task_pwm;

    // Disable memory swap
    mlockall(MCL_CURRENT|MCL_FUTURE);
    // Perform auto-init of rt_print buffers if the task doesn't do so
    rt_print_auto_init(1);

    // Passage de la boucle principale en mode temps reel
    if( rt_task_shadow( &task_main, "MAIN", HIGHEST_PRIO-10, 0 ) != 0 ) {
        rt_printf("rt_task_shadow error for MAIN\n");
        exit(EXIT_FAILURE);
    }
    // Open the device (Write permissions)
    iopl(3);
    if (ioperm(LPT, 1, 1) < 0) {
        perror("ioperm()");
        exit(1);
    }
    // Creer un sémaphore sur une ressource (implémente le PIP)
    if ( rt_mutex_create(&mut_smart,"SMARTPHONE") !=0 ) {
            rt_printf("rt_mutex_create error for SMARTPHONE\n");
            exit(EXIT_FAILURE);
    }
    // Creer une boite aux lettres pour communiquer la fréquence
    if ( rt_queue_create(   &rtmbx_freq,"FREQMBX",
                            sizeof(double)  /* pool size */,
                            1               /* qlimit */,
                            Q_FIFO ) !=0 ) {
            rt_printf("rt_queue_create error for FREQMBX\n");
            exit(EXIT_FAILURE);
    }
```



```c
    // Creer une boite aux lettres pour communiquer le rapport cyclique
    if ( rt_queue_create(   &rtmbx_rcy,"RCYMBX",
                            1*sizeof(int)   /* pool size */,
                            1               /* qlimit */,
                            Q_FIFO ) !=0 ) {
            rt_printf("rt_queue_create error for RCYMBX\n");
            exit(EXIT_FAILURE);
    }
    // Creer la tâche de communication avec le smartphone
    if (rt_task_spawn(  &task_comSmartphone /* task descriptor */,
                        "SMARTPHONETASK"    /* name */,
                        0                   /* 0 = default stack size */,
                        HIGHEST_PRIO-2      /* priority */,
                        T_JOINABLE          /* needed to call rt_task_join after */,
                        &comSmartphone      /* entry point (function pointer) */,
                        NULL                /* function argument */ )!=0) {
        rt_printf("rt_task_spawn error for SMARTPHONETASK\n");
        exit(EXIT_FAILURE);
    }
    // Creer la tâche de gestion pwm
    if (rt_task_spawn(  &task_pwm           /* task descriptor */,
                        "PWMTASK"           /* name */,
                        0                   /* 0 = default stack size */,
                        HIGHEST_PRIO        /* priority */,
                        T_JOINABLE          /* needed to call rt_task_join after */,
                        &pwm                /* entry point (function pointer) */,
                        NULL                /* function argument */ )!=0) {
        rt_printf("rt_task_spawn error for PWMTASK\n");
        exit(EXIT_FAILURE);
    }
    // Creer la tâche de traitement
    if (rt_task_spawn(  &task_traitement    /* task descriptor */,
                        "TRAITEMENTTASK"    /* name */,
                        0                   /* 0 = default stack size */,
                        HIGHEST_PRIO-1      /* priority */,
                        T_JOINABLE          /* needed to call rt_task_join after */,
                        &traitement         /* entry point (function pointer) */,
                        NULL                /* function argument */ )!=0) {
        rt_printf("rt_task_spawn error for TRAITEMENTTASK\n");
        exit(EXIT_FAILURE);
    }
    // Rejoint les tâches en attente de leur destruction
    rt_task_join(&task_comSmartphone);
    rt_task_join(&task_pwm);
    rt_task_join(&task_traitement);
    // Détruit les objets
    rt_task_delete(&task_comSmartphone);
    rt_task_delete(&task_pwm);
    rt_task_delete(&task_traitement);
    rt_queue_delete(&rtmbx_freq);
    rt_queue_delete(&rtmbx_rcy);
    rt_mutex_delete(&mut_smart);

    rt_printf("END\n");
    return 0;
}
```