ENVIRONNEMENTS POUR LE TEMPS REEL EMBARQUE REPARTI

SPECIALITE SAR, UNIVERSITE PARIS VI

Barème indicatif, **sans document**
Les téléphones portables doivent être éteints et rangés dans vos sacs.

Les **4** parties de l'examen sont indépendantes.

Il sera tenu compte de la présentation et de la clarté dans la rédaction.
Seules les réponses précises et justifiées seront considérées.

I. Chaîne croisée (~10 min – 3 pts)

Question 1
Définir les termes suivant : compilation croisée, et board support package

Question 2
On dispose du code source suivant qui a été réalisé pour le système d'exploitation RTEMS, sur MPC860.
Expliquer en quoi les lignes pointées par les flèches sur le coté droit du texte sont spécifiques à ce déploiement en particulier
(Identifier la dépendance, s'il y en a une : OS & Matérielle ou Matérielle seule).

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>                                                    ←

rtems_task init (rtems_task_argument ignored)                        ←
{
 float x =0.3;
 printf("%f", sin(x));                                               ←
  exit( 0 );
}

/* configuration information */

#define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_MAXIMUM_TASKS 1
#define CONFIGURE_INIT
#define CONFIGURE_INIT_TASK_ENTRY_POINT init
```

II. Langages Synchrones (~20min – 3 pts)

Pour décrire d'un point de vue mathématique les signaux, nous utiliserons la notation des suites numériques, soit U une suite de valeurs de type T, alors U(i) désigne la i-ème valeur de la suite.

Question 1 (1 point)
Expliquez le sens des opérateurs sur les signaux suivants: pre S ; S1 → S2

Question 2 (2 points)

On considère le programme lustre suivant :

```
node diffs(F,STEP,init: real) returns (Y: real);
        let
              Y = init -> ((F - pre(F))*STEP) ;
 tel
```

Donner la relation mathématique liant Y(n) à F(n) et F(n-1) en 0 et n>1.

Le noeud TRANSIENTLOCK implémente le comportement suivant :

```
node TRANSIENTLOCK (set: bool; delay: int ; input : real) returns (level: real);
var count: int
let
if set then count=delay ;
       else if false->(count>0) ; then pre(count)-1 else 0;

        level = 0.0 -> (if (count>0) then if set
                                                then level=input ;
                                                else input= pre(level) ;
                                        else level=0.0) ;
tel
```

Déroulez l'exécution pour les signaux d'entrée suivants (nous vous recommandons de donner les séquences de valeurs pour *count* et *level*)

set	True	False	False	True	False	True	False	False
delay	2	0	2	1	2	2	0	0
input	5.0	0.1	3.3	2.0	0.0	1.0	0.0	0.0

III. AADL (~ 30 min - 6 pts)

On considère un système de calcul de vitesse. Ce système (`Speed_Calculator`) est (entre autre) composé d'un capteur de position (`Position_Sensor`) qui envoie la position courante du système toutes les 10 millisecondes.

- Le thread `Position_Thread` reçoit cette information via son port `input` et enregistre cette donnée dans un tableau de position (`Position_array`) via l'interface d'accès aux données (`P`). Cette écriture est réalisée par un appel au sous-programme `Write`.
- Périodiquement, le thread `Speed_Thread` lie les informations contenues dans le tableau via son interface d'accès aux données (`P`). Cette lecture est réalisée par un appel au sous-programme `Read`.

En connaissant l'intervalle séparant la production de deux données, `Speed_Thread` peut alors calculer la vitesse courante du système. Nous proposons d'exécuter `Speed_Thread` à plus basse fréquence que `Position_Thread` pour que `Speed_Thread` collecte un tableau de positions mises à jour.

Nous donnons sur la page suivante le modèle AADL incomplet associé à ce système.

```
data Speed
end Speed;

data Position_Array
end Position_Array;

system Speed_Calculator
end Speed_Calculator;

system implementation Speed_Calculator.impl
subcomponents
  The_CPU : processor CPU;
  The_Sensor : device Position_Sensor;
  The_Process : process Speed_Process.impl;
connections
  --- To be completed
properties
  Actual_Processor_Binding => reference The_CPU
applies to The_Process;
end Speed_Calculator.impl;

processor CPU
properties
Scheduling_Protocol
=> Rate_Monotonic_Scheduling;
end CPU;

process Speed_Process
features
  Input: in event data port Position_Array;
end Speed_Process;

device Position_Sensor
features
  Output: out event data port Position_Array;
end Position_Sensor;

process implementation Speed_Process.impl
subcomponents
  The_Pos_Th:   thread Position_Thread.impl;
  The_Speed_Th:   thread Speed_Thread.impl;
  The_Pos: data Position_array;
connections
  cnx1 : event data port Input ->
The_Pos_Th.Input;
  cnx2 : data access The_Pos_Th.Pos -> The_Pos;
  cnx3 : data access The_Pos_Th.Pos -> The_Pos;
end Speed_Process.impl;
thread Speed_Thread
features
  Pos : requires data access Position_Array;
  Output : out event data port;
properties
  Dispatch_Protocol   => --- To be completed
  Period              => --- To be completed
  Compute_Execution_Time => 0 Ms .. 30 Ms;
end Speed_Thread;

thread Position_Thread
features
  Pos : requires data access Position_Array;
  Input : in event data port Position_Array;
properties
  Dispatch_Protocol   => --- To be completed
  Period              => --- To be completed
  Compute_Execution_Time => 0 Ms .. 5 Ms;
end Position_Thread;

thread implementation Position_Thread.impl
calls
{
 --- To be completed
}
connections
    --- To be completed
end Position_Thread.impl;

thread implementation Speed_Thread.impl
calls
{
 --- To be completed
}
connections
 --- To be completed
end Speed_Thread.impl;

subprogram Write
feature
  Pos: requires data access Position;
end Write;

subprogram Read
feature
  Pos: requires data access Position;
end Read;
```

Question 1 (1,5 points)

Compléter la section *properties* des composants `Position_Thread` et `Speed_Thread`.

Question 2 (2 points)

Compléter la section *connections* des composants `Speed_Calculator.impl`, `Speed_Thread.impl` et `Position_Thread.impl`.

Question 3 (1,5 points)

Compléter la section *calls* des composants `Speed_Thread.impl` et `Position_Thread.impl`.

Question 4 (1 point)

Expliquer brièvement pourquoi il n'est pas nécessaire d'associer une priorité aux composants `Speed_Thread.impl` et `Position_Thread.impl` pour connaître leurs priorités relatives.

IV. Problème - Noyaux temps réels (~1h – 8 pts)

Nous allons nous intéresser à l'étude du déploiement d'un lot de tâches sur un support d'exécution ne proposant pas de service d'ordonnancement en-ligne.

Ainsi, le **modèle de tâche** implémenté au niveau du **support d'exécution *X*** est le suivant :

- Les tâches ne peuvent pas être préemptées : l'ordonnancement est spécifié statiquement à travers une séquence d'instances de tâches à exécuter sans possibilité d'interrompre une tâche pour en lancer une autre.
- L'exécution de cette séquence est relancée périodiquement avec une période HP qui définit un cycle d'exécution.
- En plus de l'ordre d'exécution, la date d'activation de l'instance peut être spécifié pour chaque instance de tâche.
- L'enchaînement des instances de tâches à exécuter est mémorisé dans une liste chainée de descripteurs d'instance de tâche (TLI). Chaque descripteur contient :
  - Le nom de la tâche,
  - La date d'activation de l'instance par rapport au début du cycle courant. Si cette date n'est pas dépassée le système se met en pause jusqu'à cette date (le système et non la tâche, i.e. aucune autre tâche ne s'exécute entre temps).
  - Et la prochaine tâche à exécuter (rien si l'activité du cycle courant est terminée).
- A chaque tâche est associé un descripteur contenant le nom de la tâche, et une référence vers la fonction à exécuter à chaque exécution d'une instance de la tâche. Par exemple, considérons une tâche T_1 exécutant périodiquement la fonction f, alors $D=\{T_1,f\}$.Tous ces descripteurs sont stockés dans TL, la liste des descripteurs de tâches.

Le but du problème est d'aborder le question du déploiement sur un tel système d'un lot de tâches périodiques temps réels selon un ordonnancement hors-ligne (avec des tâches non-préemptibles dans un premier temps)

Convention de notation

On notera T_k^i la i-ème activation de la tâche k. Par exemple, il existe plusieurs ordres d'exécution possibles des trois premières instances de deux tâches T_1 et T_2, on pourra utiliser cette notation deux instances successives d'une même tâche lors de la description d'une exécution du système: e.g. T_2^1, T_1^1, T_2^2 ou T_1^1, T_2^1, T_2^2 .

Illustration du problème

Soit T1 une tâche de période 1 exécutant la fonction f de WCET W1=0,2, puis T2 une tâche de période 2 exécutant la fonction g de WCET W1=0,6.

Un déploiement sur le support d'exécution *X* de ces deux tâches donne la liste chainée suivante (le champ pointant vers la tâche suivante est masqué) :

$$TLI = (T_1,0) \rightarrow (T_2,0) \rightarrow (T_1,1) \text{ et } TL = \{(T_1,f),(T_2,g)\}$$

Le chronogramme résultant est le suivant (hypothèse temps d'exécution = WCET) :

[0,0.2]	]0.2, 0.8]	]0,8,1]	]1,1.2]	]1.2, 2[
T_1^1	T_2^1	rien	T_1^2	rien

Soient les tâches T_1 et T_2 telles que :

	Période	WCET	fonction
T_1	0,33	0,2	h
T_2	0,2	0,05	g

RAPPEL : les tâches ne sont pas préemptées !

Questions

1) Supposons que le système atteint l'instant t0+0,33 et t0+0,4 avec t0 le début d'un cycle d'exécution. Quelles instances de tâches doivent être complétées à ces dates respectives?
2) Même question en t0+0,66 et t0+0,6
3) En déduire une affectation des différents descripteurs du système permettant de vérifier la contrainte Deadline = Période, pour chaque activation d'une tâche. (indice : Déterminez d'abord l'ordre d'exécution des instances de tâches)
4) Fournissez le chronogramme représentant l'exécution du système configuré selon vos descripteurs avec l'hypothèse : temps d'exécution == pire temps d'exécution
5) En supposant que l'on ne modifie pas la configuration des descripteurs, donnez le nouveau chronogramme d'exécution en supposant WCET de h (i.e. WCET de T1) égal à 0,23 au lien de de 0,2.

Remarque : le « mapping » du modèle de tâche théorique vers le modèle « système » n'a de sens que si les paramètres sont parfaitement connus et ne varient pas.

Adaptation du lot de tâches au support d'exécution

La dernière question permet de voir une méthode classique pour rendre ordonnançable un lot de tâches sur un tel système en relâchant l'hypothèse sur la non préemption. Dans le cadre de l'ordonnancement hors-ligne, la préemption signifie simplement que l'on peut décomposer une tâche en une séquence de traitements t1, t2, …., tn de tel sorte à exploser cette tâche en n sous tâches, chacune prenant un traitement en charge, au moment de la définition de l'ordonnancement hors ligne sur le support d'exécution.

On suppose que l'implémentation de h vérifie les deux points suivants :

- L'implémentation de h correspond à exécuter séquentiellement *ha* et *hb* de telle sorte que *hb* prend en entrée la sortie de *ha* (précédence stricte entre les deux traitements)
- WCET(*ha*) = 0,2 et WCET (*hb*)=0,03 (compatible avec (WCET (h) = 0,23

Question

6) Expliquer comment l'éclatement de T1 en deux tâches T1a et T1b permet d'instancier les descripteurs du système de sorte à produire un système respectant la contrainte Période = Deadline des tâches