Module STR
Examen session 1, 19 décembre 2013
Systèmes temps réel
Durée : 1h20
Tous documents autorisés

Singhoff Frank
singhoff@univ-brest.fr
C-202

NB : ce document contient 2 feuilles recto/verso.

Dans ce sujet, les deux exercices traitent du même système. Il est donc fortement conseillé de lire l'intégralité du sujet avant de traiter l'un de ces exercices.

Exercice n^o 1 : Ordonnancement temps réel (7 points)

On regarde dans cet exercice un système de mesure embarqué dans un avion et composé d'un ensemble de tâches définies par les paramètres suivants :

Nom	Capacité	Période
Affichage	3	30
Capteur1	1	5
Capteur2	1	5
Analyse	1	10

Les tâches $Capteur1$ et $Capteur2$ mesurent la temperature à deux endroits de l'avion. La tâche *Analyse* contrôle ces températures et calcule leur moyenne. La tâche $Affichage$ affiche à l'écran la température moyenne. On suppose les tâches indépendantes. Ces tâches sont ordonnancées selon un algorithme à priorité fixe preemptif. On suppose un ordonnancement POSIX dont les priorités varient de 0 à 31 (0 est la priorité la plus forte).

Questions :

1. Proposez une priorité pour chaque tâche selon la politique Rate Monotonic. Vous proposerez un niveau de priorité différent pour chaque tâche.
2. Vérifiez que ce jeu de tâche est ordonnançable sans dessiner le chronogramme d'ordonnancement des tâches.
3. Vérifiez le résultat de la question précédente en dessinant sur la période d'étude l'ordonnancement des tâches.
4. On souhaite augmenter le nombre de capteur sur l'avion. Combien de capteurs supplémentaires est il possible d'ajouter dans ce système tout en respectant les échéances de toutes les tâches ?
5. En pratique, les tâches se sont pas indépendantes : elle communiquent via une donnée partagée qui est protégée par un sémaphore. Toutes les tâches verrouillent le sémaphore au début de la dernière unité de temps de leur capacité et libère le sémaphore à la fin de la dernière unité de temps de leur capacité : les sections critiques ont donc une durée de 1 unité de temps. On suppose que l'on utilise PIP. Vérifier si le jeu de tâche reste ordonnançable même avec cette ressource partagée.

Exercice n^o 2 : Programmation concurrente avec Ada (7 points)

Dans cette partie, on regarde une mise en oeuvre de notre système de mesure en Ada. Le système de mesure est implanté par le paquetage Ada *mesure*.

Le paquetage *mesure* est donné en annexe. Il propose un type pour les tâches $Capteur$, *Analyse* et $Affichage$. Dans ce paquetage, les mesures effectuées sont des mesures de la temperature de l'avion. On suppose données les fonctions :

- *lire_donnee* qui permet de lire un capteur.
- *analyse_donnees* qui effectue diverses analyses sur les températures et retourne la température moyenne.

Le comportement des tâches est le suivant :

- Une tâche *Capteur* doit d'abord recevoir un identifiant unique grâce à l'entrée *recevoir_id*. Puis, dans une boucle, elle doit lire toutes les secondes la température (grâce à la fonction *lire_donnee*) et sauvegarder la valeur obtenue dans le tableau *donnee_capteur_analyse*. L'identifiant unique de la tâche permet aux différentes tâches de sauvegarder la température à une position différente dans le tableau.
- Grâce à la fonction *analyse_donnees*, la tâche *Analyse* calcule toutes les secondes une moyenne des températures sauvegardées dans le tableau. Le résultat de ce calcul est mémorisé dans la variable partagée *donnee_analyse_affiche*.
- Enfin, la tâche *Affiche* doit afficher à l'écran toutes les secondes la valeur de *donnee_analyse_affiche*.

Questions :

1. Le corps du paquetage *mesure* contient une erreur qui l'empêche d'être compilé correctement. Décrivez cette erreur et corrigez le paquetage.
2. Proposez une procédure principale qui déclare deux tâches *Capteurs*, une tâche *Affiche* et une tâche *Analyse*. La procédure principale doit communiquer un identifiant unique à chaque tâche capteur en utilisant l'entrée *recevoir_id*.
3. Le corps du paquetage *mesure* contient deux erreurs de synchronisation des tâches qui l'empêcheront de fonctionner correctement. Décrivez ces erreurs et corrigez le paquetage.
4. La mise en oeuvre actuelle du paquetage *mesure* est difficilement réutilisable pour d'autres types de mesure. On souhaite paramétrer le paquetage afin de pouvoir l'instancier pour un nombre quelconque de capteurs et pour un type de donnée autre que *temperature*. Proposez une version générique du paquetage *mesure*.

Annexe

```
package mesure is

  type temperature is new integer range 0..100;

  nb_capteurs  : integer :=2;

  type table_donnee is array (1..nb_capteurs) of temperature;
  donnee_capteur_analyse : table_donnee;
  donnee_analyse_affiche : temperature;

  task type capteur is
    entry recevoir_id (id : in integer);
  end capteur;
  task type analyse;
  task type affiche;

end mesure;
-----------------------------------------
package body  mesure is

  function lire_donnee return temperature is begin ...
  function analyse_donnees(t : in table_donnee)
    return temperature is begin ...


  task body capteur is
  begin
      accept recevoir_id(id : in integer) do
          loop
            donnee_capteur_analyse(id):=lire_donnee;
            delay 1.0;
          end loop;
      end recevoir_id;
  end capteur;

  task body analyse is
  begin
    loop
      donnee_analyse_affiche
        :=analyse_donnees(donnee_capteur_analyse);
      delay 1.0;
    end loop;
  end analyse;

  task body affiche is
  begin
    loop
      put(donnee_analyse_affiche);
      delay 1.0;
    end loop;
  end affiche;

end mesure;
-----------------------------------------
```