Rapport d'exécutifs temps-réel
- Transfert de messages -

École Polytech de Nantes
Département Électronique et Technologie du Numérique

Rédigé par
Mathis BRIARD
Victor DUFRENE

Professeur encadrant :
Olivier PASQUIER

Table des matières

Table des figures

Liste des tableaux

Résumé

L'exécutif temps-réel de systèmes électroniques est une discipline essentielle lorsqu'il s'agit, dans le cadre d'une future carrière d'ingénieur, de réaliser un produit répondant à des contraintes de temps. Et ce domaine est d'autant plus indispensable dans un monde où les systèmes numériques sont de plus en plus complexes et rapides, demandant de réaliser des tâches dans une fenêtre de temps bien déterminé. Ce rapport s'inscrit dans la présentation d'une application affichant l'heure sur un écran.

1 Introduction

1.1 Contextualisation

La formation ETN (Électronique et technologies numériques) offerte par l'école polytechnique de l'Université de Nantes propose d'aborder diverses branches de l'électronique, du traitement du signal au systèmes à microprocesseur en passant par l'électronique analogique des hautes-fréquences. Cet ensemble de domaines techniques nécessite des compétences en matière de méthodologie de conception. Ce rapport s'inscrit dans la conception d'un appareil de marquage routier avec la méthode MCSE. La méthode MCSE (Méthode de conception des systèmes électroniques), née à Ireste par l'impulsion de Jean-Paul Calvez, cette méthode a été implantée au sein d'un outil nommée CoFluent rachetée par Intel® depuis 2011. Cette méthode fait désormais partie de la culture de la formation et constitue l'outil de conception premier de l'ingénieur ETN.
Ce rapport se décompose en diverses parties. Il s'agira dans un premier temps de rappeler le cahier des charges de la conception de cet appareil de marquage routier. Dans un second temps la partie spécification sera traitée et pour finir il s'agira de parler de la conception.

1.2 Objectifs

Ce rapport vise à retranscrire les résultats du travail pratique réalisé sur les files (transferts) de messages. Les objectifs des manipulations étaient d'utiliser le mécanisme de boîte aux lettres puis d'estimer l'utilisé de l'exclusion mutuelle.

2 Situation

La situation que nous visons à implémenter est une file de message qui permet la communication inter-processus. L'application considérer est celle d'un système qui affiche l'heure dont la structure fonctionnelle est donné à la figure 1.

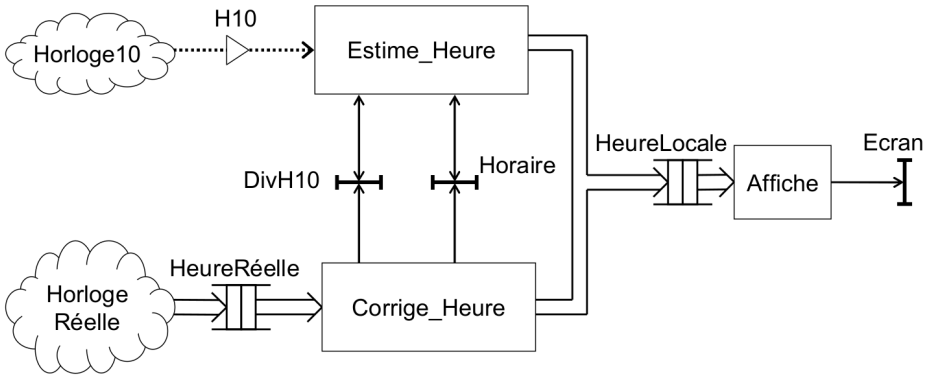

Figure 1: Structure fonctionnelle de l'application

L'application consiste à afficher sur Ecran, l'heure (heure, minute, seconde). L'heure peut être obtenue par estimation à partir d'une horloge à 10Hz (H10) et pour éviter les dérives, l'heure réelle est reçue par l'intermédiaire de HeureRéelle (heure, minute, seconde). Pour information, afin de limiter la longueur des observations, la base de temps de l'environnement est beaucoup plus rapide que la seconde.

Fonction Estime_Heure

Action activée sur H10, événement périodique apparaissant 10 fois par seconde. Elle incrémente la variable DivH10 si elle est inférieure à 9, sinon elle met la variable à 0 et la variable Horaire est incrémentée d'une seconde. Un message est transmis dans HeureLocale. Le message indique que la valeur transmise est une valeur estimée et contient la nouvelle valeur de Horaire.

Fonction Corrige_Heure

Action activée sur HeureRéelle. Elle copie la valeur reçue dans Horaire, met la variable DivH10 à 0 et transmet un message dans HeureLocale. Le message indique que la valeur transmise est une valeur réelle et contient la nouvelle valeur de Horaire.

Fonction Affiche

Cette fonction copie sur Ecran l'heure reçue par l'intermédiaire de HeureLocale en indiquant s'il s'agit de l'heure estimée ou de l'heure réelle (c'est la seule tâche ou vous pouvez faire un printf).

Le réseau de Pétri qui décrit le comportement des fonctions `Estime_Heure` et `Corrige_Heure` est donné à la figure 2 ci-dessous.

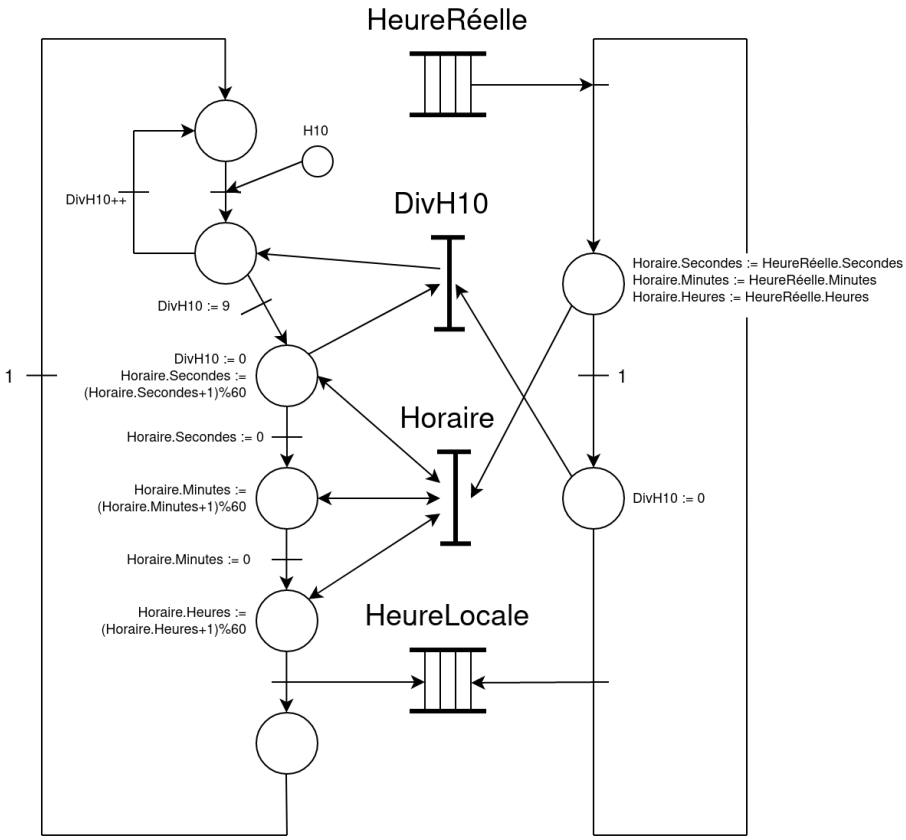

Figure 2: Réseau de pétri du comportement des fonctions `Estime_Heure` et `Corrige_Heure`

Dans ce réseau on est en présence de deux variables partagées `Div10` et `Horaire`. Une question logique à ce poser est : doit on mettre des sémaphores pour éviter des exclusions mutuelles ?

La réponse est non car l'environnement est synchrone. Les tâches `Estime_Heure` et `Corrige_Heure` sont prêtes en même temps puis s'activent en même temps. Cela implique une gestion simple des priorités. C'est pourquoi il est possible d'éviter la mise en place de sémaphore.

3 Solution classique

Cette partie vise à mettre en place la solution décrite précédemment. Pour l'implantation sur VxWorks, il est nécessaire d'utiliser les éléments suivants :

- une tâche `Estime_Heure`;
- une tâche `Estime_Heure`;
- une tâche `Estime_Heure`;
- un sémaphore `H10`;
- une file de message `FMHeureLocale`;
- une file de message `FMHeureReelle`;
- une variable partagé `DivH10`;
- une variable partagé `horaire`.

On précise que la variable `horaire` est d'un type prédéfini `type_heure` qui contient les secondes, les minutes et les heures dans une structure.

Pour le bon fonctionnement du programme, il est nécessaire d'attribuer des priorités aux différentes tâches.Dans l'environnement VxWorks, la tâche de priorité la plus élevée est celle qui dispose du numéro de priorité le plus faible. La tâche de priorité 0 est la plus prioritaire et la tâche de fond est la tâche de priorité 255. On évite d'attribuer les priorités comprises entre 0 et 10, réservée à l'environnement VxWorks. Les priorités suivantes sont affectées aux tâches :

Tâche	Priorité
`Estime_Heure`	13
`Corrige_Heure`	14
`Affiche`	15

Table 1: Attribution des priorités aux différentes tâches

Le code commenté de la solution à un utilisateur pour les priorités ci-dessus est disponible en Annexe 1. La simulation de la situation est lancée et analysée ci-dessous.

3.1 Validation du comportement

La figure 3 montre la sortie de la console pour une exécution classique du programme.

```
value = 0 = 0x0
-> start
Demarrage de l'estimation de l'horloge
Demarrage de la correction de l'horloge
Demarrage de l'affichage
Fin de start
value = 0 = 0x0
-> Origine du message : Corrigee
l'Heure est: 0 Heure, 1 min, 2 sec
Origine du message : Corrigee
l'Heure est: 0 Heure, 2 min, 3 sec
Origine du message : Corrigee
l'Heure est: 0 Heure, 2 min, 4 sec
Origine du message : Corrigee
l'Heure est: 0 Heure, 2 min, 5 sec
Origine du message : Estimee
l'Heure est: 0 Heure, 2 min, 6 sec
Origine du message : Corrigee
l'Heure est: 1 Heure, 2 min, 3 sec
Origine du message : Corrigee
l'Heure est: 1 Heure, 3 min, 4 sec
Origine du message : Corrigee
l'Heure est: 1 Heure, 3 min, 5 sec
Origine du message : Corrigee
l'Heure est: 1 Heure, 3 min, 6 sec
Origine du message : Estimee
l'Heure est: 1 Heure, 3 min, 7 sec
```

Figure 3: Sortie sur la console

Une fois l'initialisation des tâches effectués dans le `start`, nous observons un affichage de l'heure provenant des deux tâches. On note que l'heure affiché provient beaucoup plus souvent de `Corrige_Heure` que de `Estime_Heure`. La figure 3 montre la simulation temporelle des tâches. On y observe l'évolution des états des différentes tâches au cours du temps ainsi que l'influence des divers événements tels que les sémaphores ou les files de messages.

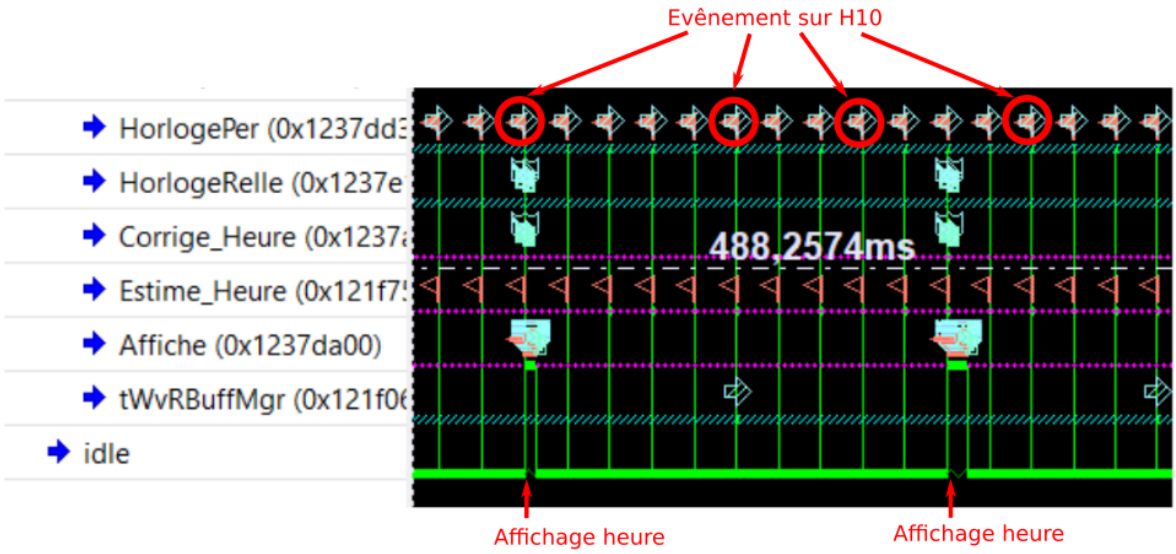

Figure 4: Simulation l'impact de la tâche `Estime_Heure` et `Corrige_Heure`

On précise que les tâches `HorlogePer` et `HorlogeRelle` font toutes deux parties de l'environnement du système. La première donne gère le sémaphore H10 tandis que la deuxième gère la file de message HeureRéelle.
Comme attendu par le comportement spécifié par le cahier des charges. On ici que l'affichage de l'heure (par la tâche `Affiche`) se fait tous les dix événements sur H10. Le comportement observer ici est donc uniquement lié à la tâche `Estime_Heure`. Pour observer l'influence de la seconde tâche, attardons nous sur la figure 5 ci-dessous.

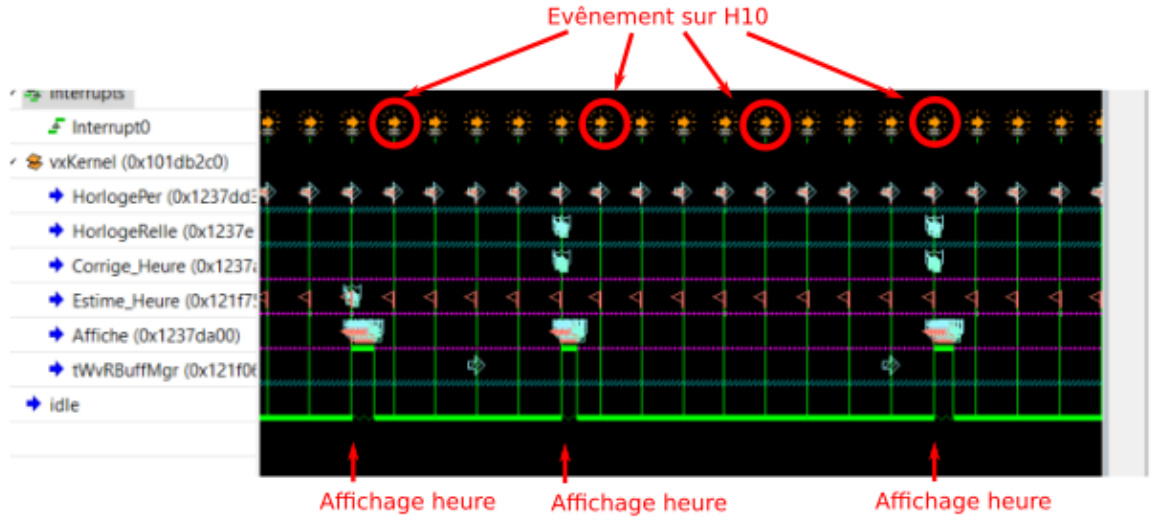

Figure 5: Simulation l'impact de la tâche `Corrige_Heure`

La première chose que l'on observe ici est la présence de trois affichage de l'heure et que les deux premiers sont séparé de moins de dix coups d'horloges. Si l'on zoome sur l'endroit où le deuxième affichage de l'heure s'effectue nous obtenons la figure 6.

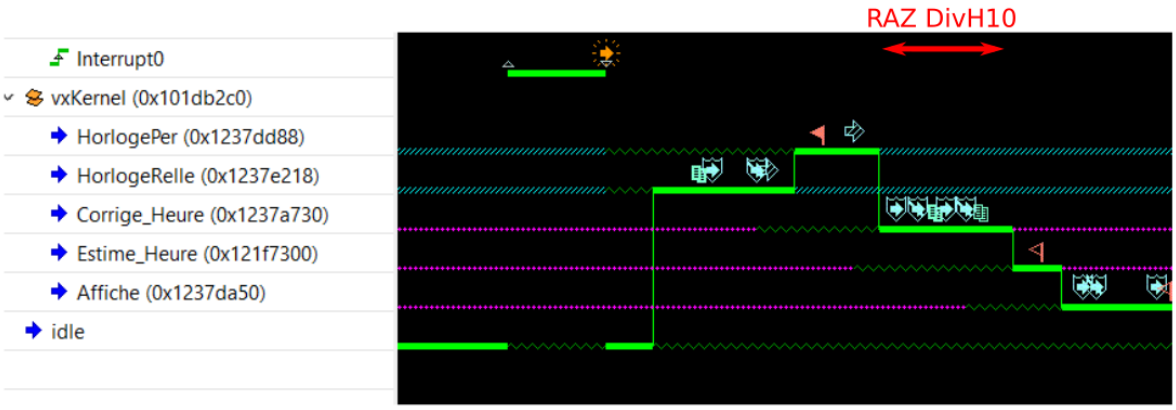

Figure 6: Zoom sur le 2e affichage de la figure 5

On observe sur la figure ci-dessus l'activation de `HorlogeRelle`. Cette tâche va envoyer une nouvelle donnée sur la file de message HeureRéelle ce qui "dit" va dire à la tâche `Corrige_Heure` de s'exécuter. Ayant une priorité supérieure à `Estime_Heure`, `Corrige_Heure` va s'exécuter en premier puis mettre à 0 le compteur DivH10 comme prévu. La tâche `Estime_Heure` va ensuite s'exécuter et donc incrémenter le compteur DivH10. C'est pourquoi on ne compte que 9 événements sur H10 entre le deuxième affichage de l'heure et le troisième sur la figure 5 contrairement à la figure 4 où il y a 10 événements.

3.2 Asservissement de la boîte aux lettres

La section qui suit s'intéresse au principe de l'asservissement de la boîte aux lettres. Il s'agit de comprendre le fonctionnement du système lorsque la file de message est pleine. Le cas de la file de message `HeureLocale` vide n'est pas considérée ici de part sa simplicité. En effet si celle-ci est vide alors la tâche `Affiche` appelle la fonction bloquante `msgQReceive` et attend l'émission d'un nouvel horaire. La suite abordera donc le cas de la file de message pleine. Afin d'assurer un remplissage complet de la file de message `HeureLocale`, le programme est modifié pour faciliter l'arrivée de cet événement. La taille de la file de message est réduite à 3 et une fonction de VxWorks nommée `taskDelay` est utilisée. Celle-ci prends en argument un entier correspondant au nombre de 'top' horloge du CPU virtuel à attendre avant de continuer la suite de la tâche `Affiche`. La durée d'exécution de la tâche `Affiche` est alors augmentée, ce qui laissera le temps à `Estime_Heure` et `Corrige_Heure` de remplir la file de message. Le choix est arbitraire et se porte sur 500 ms. Afin de déterminer le nombre de 'top' horloge à compter, la fonction `sysClkRateGet` est rentrée dans la console de VxWorks et renvoie la fréquence du CPU virtuel, ici 60 Hz. Le produit 60×0.5 donne 30 'top' horloge.
La figure ci-dessous représente le diagramme d'activité global du système.

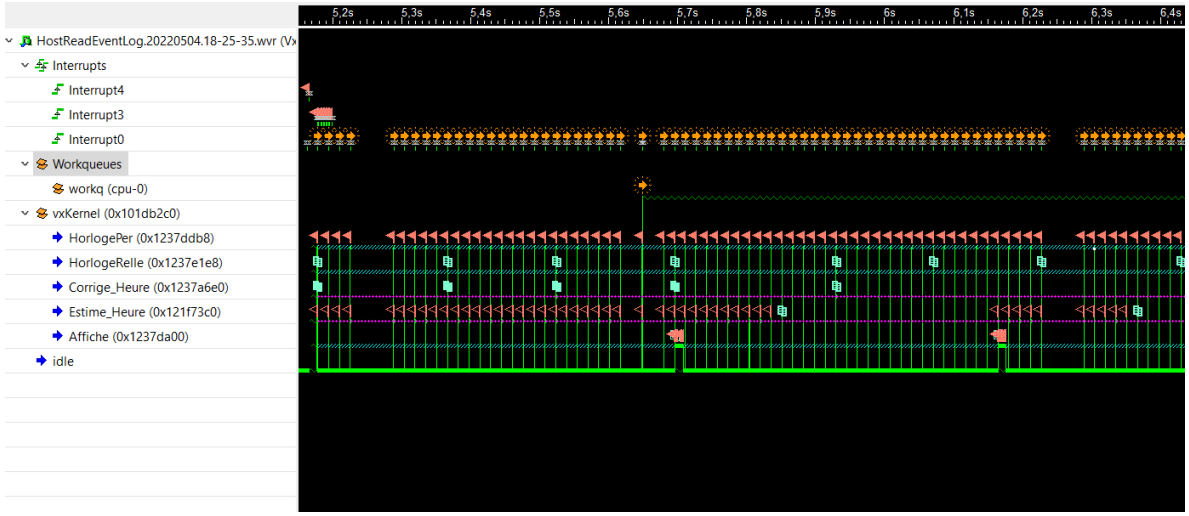

Figure 7: Vue globale de la simulation

La simulation effectuée regroupe l'ensemble des tâches du système comme `Estime_Heure`, `Corrige_Heure` et `Affiche`. Celle-ci regroupe également le comportement temporel de l'environnement avec `HorlogePer` et `HorlogeRéelle` qui sont respectivement les entités `Horloge10` et `HorlogeRéelle`. Parmi cet ensemble d'éléments, les plus en haut sont les plus prioritaires et ceux qui sont les plus en bas sont les moins prioritaire.[1]
Sur la ligne `Estime_Heure`, la tâche récupère convenablement le sémaphore `H10` et arrive un moment où celui-ci n'est plus pris. Cet événement témoigne du remplissage complet de la file de message `HeureLocale`. Une analyse étape par étape démontre ce phénomène.

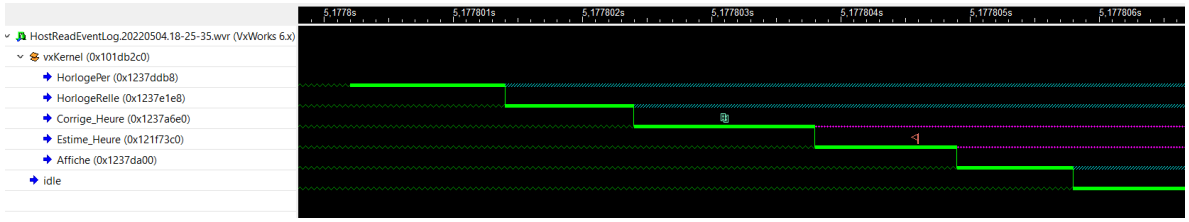

Figure 8: Initialisation des tâches

La figure ci-dessus représente l'initialisation des tâches avant de rentrer dans les boucles infinies. Dans le cas de `Corrige_Heure`, celle-ci s'initialise et rentre dans la boucle infinie, puis appelle la fonction `msgQReceive` et reste bloquée dans l'attente d'une donnée dans la file de message `HeureRéelle`.
En ce qui concerne `Estime_Heure`, cette tâche s'initialise et rentre dans la boucle infinie pour au final être bloquée car aucun sémaphore `H10` a été émis. Par la suite, la figure suivante représente le premier envoi d'une donnée dans la file de message `HeureRéelle`. L'appel de la fonction `msgQSend` est effectué de la part de `HorlogeRéelle`. Le noyau temps-réel donne la main à `Corrige_Heure` qui appelle `msgQSend`, en effet à l'initialisation celle-ci est restée bloquée avec `msgQReceive`, l'envoi d'une donnée dans la file de message a permis à cette tâche d'être de nouveau prête. La donnée est donc envoyée dans la file de message `HeureLocale` et `Corrige_Heure` reboucle et revient à nouveau dans l'attente d'une donnée, ici remarquable par l'appel de la fonction `msgQReceive`. La tâche `Affiche` n'est pas présente, ce qui dans une utilisation normale devrait être le cas. En regardant les temps, l'initialisation de `Affiche` se termine à 5,177 ms environ. L'envoi d'un premier message est effectué vers 5,178 ms. Seulement une microseconde sépare ces deux événements, la tâche `Affiche` est bloquée par la fonction `taskDelay` pendant une demie-seconde, avant l'appel de la fonction `msgQReceive`. Le message sera donc récupéré seulement vers 5,7 ms.

[1] Pour remarque, il est déconseillé d'attribuer une priorité supérieure à 100 pour une tâche de fond. En effet dans ce cas la tâche `RingManager` ne sera jamais exécutée ce qui peut provoquer le plantage de VxWorks.

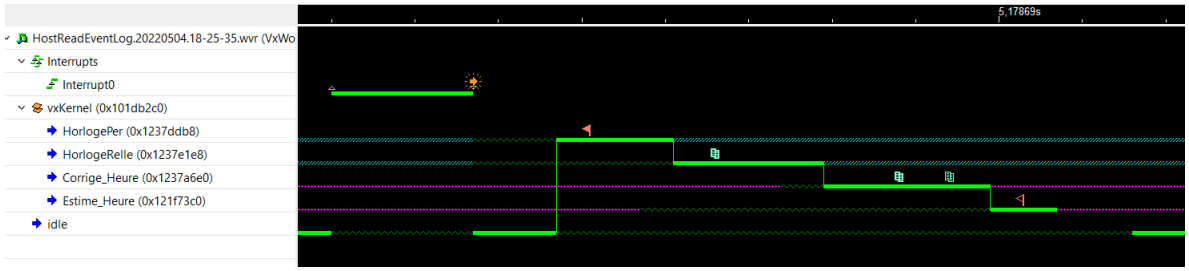

Figure 9: Premier envoi d'une donnée dans la file de message `HeureRéelle`

Il est possible de remarquer qu'entre temps, `HorlogePer` a émis le sémaphore `H10` qui a ensuite été récupéré par `Estime_Heure`. Aucun appel de `msgQSend` est effectué, il est donc possible de déduire que la variable partagée `DivH10` est inférieure à 10. Ce même comportement est visible de nombreuses fois, la figure suivante le précise.

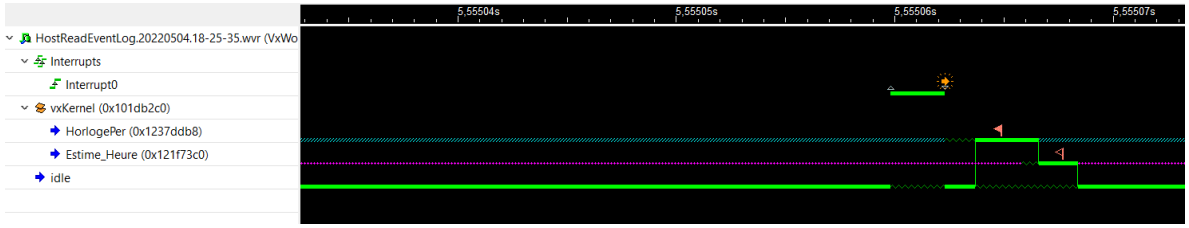

Figure 10: Émission du sémaphore `H10` et prise de celui-ci par `Estime_Heure` dans le cas où la variable partagée `DivH10` est inférieure à 10

D'après la figure (7), entre l'initialisation et jusqu'à 5,6 ms environ, 3 `msgQSend` sont effectués de la part de `HorlogeRéelle`. Ces 3 messages sont récupérés par `Corrige_Heure` et envoyés dans la file de message `HeureLocale`. Celle-ci est alors pleine.
La figure suivante représente le moment où `Affiche` sort de la fonction bloquante `taskDelay`. Avant cela un message est envoyé dans la file de message `HeureRéelle`, la tâche `Corrige_Heure` sort de la fonction bloquante et appelle ensuite la fonction `msgQSend` à 5,696205 ms. Cet appel ne signifie pas que le message est envoyé dans la file de message, en l'occurrence la tâche `Corrige_Heure` reste bloquée car la file de message `HeureLocale` est pleine.

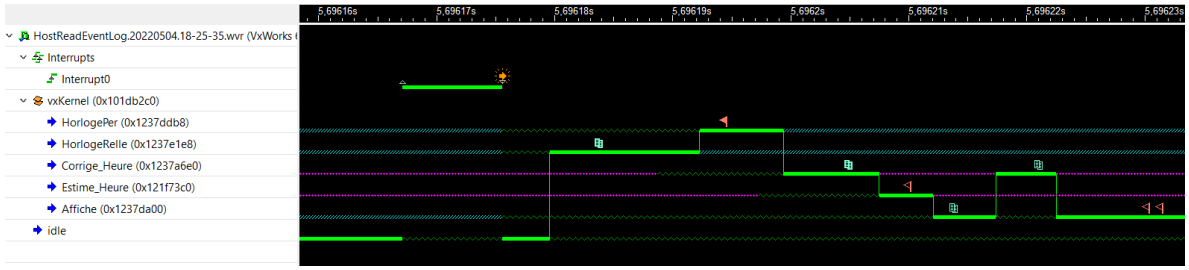

Figure 11: Réception et affichage du premier message contenu dans `HeureLocale`

À environ 5,696215 ms, la tâche `Affiche` est libérée de la fonction `taskDelay` et récupère le premier message. La file de message passe à 2 éléments. Mais celle-ci repasse immédiatement à 3 messages. En effet comme expliqué ci-dessus, la tâche `Corrige_Heure` est bloquée dans la fonction `msgQSend` dans l'attente d'une place dans la file de message. Pour preuve, la suite du diagramme montre que `Corrige_Heure` appelle la fonction `msgQReceive`. En ce qui concerne `Affiche`, celle-ci redevient bloquante à cause de l'appel de `taskDelay`, et ceci pour encore 500 ms.

La file de message `HeureLocale` est à ce moment là composée de 3 messages, celle-ci est donc remplie. La figure ci-dessous montre lorsque la variable partagée `DivH10` vaut au moins 10, ce qui n'a jamais

été le cas si l'on compte sur la figure (7) l'ensemble des `semGive` de `HorlogePer`. À ce moment là `Estime_Heure` est enfin amenée à envoyer un message dans `HeureLocale`. Cela est tenté par l'appel de la fonction `msgQSend` contenue dans `Estime_Heure`. Cependant la file de message `HeureLocale` est pleine. La tâche `Estime_Heure` est alors bloquée par `msgQSend`.

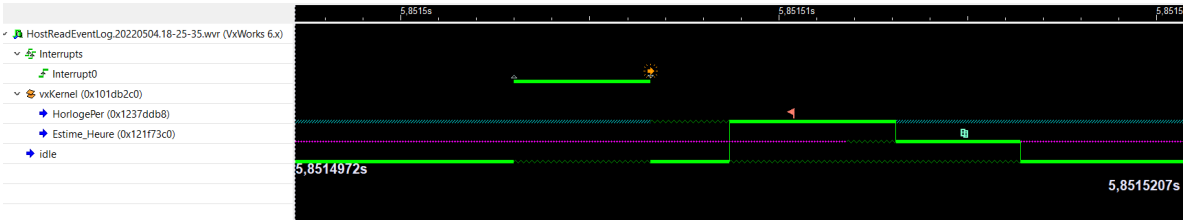

Figure 12: Première tentative d'envoi d'un message dans `HeureLocale` par `Estime_Heure`

Cela permet d'expliquer pourquoi sur la figure (7) il n'y a une absence de drapeaux `semTake` de la part de `Estime_Heure`. En effet, celle-ci est bloquée par `msgQSend`, et ceci jusqu'à ce que `Affiche` soit libérée du `taskDelay`. Également sur la figure (7) il est possible de remarquer qu'un message est envoyé dans `HeureRéelle`. Celui-ci est récupéré et `Corrige_Heure` tente de l'envoyer mais `HeureLocale` est pleine. Ce qui montre qu'ensuite tous les messages placé dans `HeureRéelle` en pourront pas être récupéré par `Corrige_Heure` car cette tâche est également bloquée par `msgQSend`.

Pour conclure sur cette section, il a été vu le comportement du système lorsque la file de message `HeureLocale` est pleine. Les contraintes temps-réel sont nullement respectées lorsque la tâche `Affiche` est moins cadencée que ces compères `Estime_Heure` et `Corrige_Heure`. Il est alors impératif d'avoir une tâche `Affiche` très réactive afin de vider le plus rapidement possible la boîte aux lettres.

3.3 Différences liées au type de sémaphore

On cherche ici à montrer la différence de comportement entre un sémaphore de type booléen ou de "type compteur". Pour utiliser un type booléen on déclarera le sémaphore avec la fonction `semBCreate` tandis que pour utiliser un "type compteur" on utilisera `semCCreate`. Pour observer des différences, on ajoutera une attente de quelques millisecondes avec la fonction taskDelay dans la tâche `Estime_Heure`. Ce ralentissement dans son exécution va faire que la tâche va accumuler les jetons en entrée.

La figure 13 montre le comportement du programme avec un sémaphore booléen.

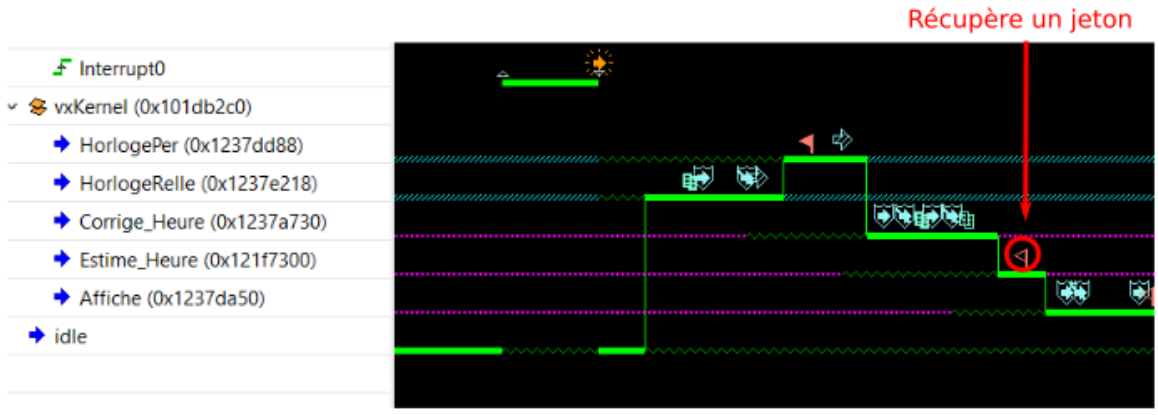

Figure 13: Simulation avec un sémaphore booléen

Ce que l'on observe ici c'est que lors de l'exécution de la tâche `Estime_Heure`, elle consomme un jeton et ceux, qu'importe le nombre de jeton qu'elle a loupée. Le cas d'un sémaphore de "type compteur" est illustré à la figure 14 ci-dessous.

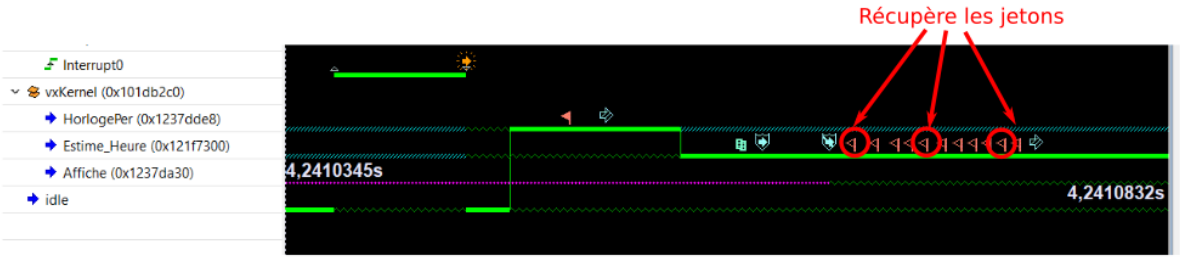

Figure 14: Simulation avec un sémaphore compteur

On observe ici que lors de son exécution, la tâche `Estime_Heure` va consommer tous les jetons qu'elle aura loupée.

4 Variante

Le système conçu est capable d'estimer l'heure à l'aide du bloc `Estime_Heure`. Cela est réalisable à l'aide du sémaphore `H10`, envoyé par l'entité `Horloge10`. Dans le cas où, pour quelconque raison, l'entité `Horloge10` n'effectue plus son rôle, la situation est critique et le système n'est plus capable d'estimer l'heure.

5 Conclusion

6 Annexe

```
/*
TP Executif temps-reel manipulation 3 : Transfert de messages
Mardi 11 avril 2022
BRIARD Mathis & DUFRENE Victor
ETN4 G2
*/

/*
Elements VxWorks requis par l'application :
- 3 taches (Estime_Heure, Corrige_Heure et Affiche)
- 1 semaphore (H10)
- 2 files de messages (FMHeureReelle et FMHeureLocale)
- 2 variables partagees (DivH10 et Horaire)
*/

/*inclusion des bibliotheques*/
// #include "vxWorks.h"
// #include "semLib.h"
// #include "taskLib.h"
// #include "msgQLib.h"
// #include "TP3_Environnement.h"
#include "string.h"
#include "stdio.h"
#include "time.h"


#define STACK_SIZE (size_t)20000
#define DivH10MAX 9
```

```
/*Declarations des semaphores*/
SEM_ID H10;

/*Declarations des files de messages*/
MSG_Q_ID FMHeureReelle;
MSG_Q_ID FMHeureLocale;

/*Declaration des TID*/
TASK_ID tidEstime_Heure;
TASK_ID tidCorrige_Heure;
TASK_ID tidAffiche;

/*Declarations des types*/
typedef struct  {
        int seconde;
        int minute;
        int heure;

} Horaire_t;

typedef struct{
        char provenance[10];
        Horaire_t horaire;
} HeureLocale_t;


/*Declaration des variables partagees*/
int DivH10;
Horaire_t horaire;

/*-----------------------------------------*/
/*---------------Fonctions-----------------*/


/*-------------Estime Heure-------------*/
void Estime_Heure(void)
{
        //Declaration des variables locales
        HeureLocale_t Heure_Locale;
        //Attente du semaphore d'eveil et procedure
        while(1)
        {
                semTake(H10, WAIT_FOREVER);
                if(DivH10<DivH10MAX)
                {
                        DivH10++;
                }else{
                        DivH10=0;
                        (horaire.seconde++)%60;
                        if(horaire.seconde==0){
                                (horaire.minute++)%60;
                                if(horaire.minute == 0){
                                        (horaire.heure++)%24;
                                }
                        }
```

```
                        char chaine[]="Estimee";
                        strcpy((Heure_Locale.provenance),chaine);
                        Heure_Locale.horaire=horaire;
                        msgQSend(FMHeureLocale, &Heure_Locale, sizeof(
                        HeureLocale_t), WAIT_FOREVER, MSG_PRI_NORMAL);
                }
        }
}

/*----------Corrige Heure----------*/
void Corrige_Heure(void)
{
        HeureLocale_t Heure_Locale;
        Horaire_t HeureReelle;
        while(1)
        {
                msgQReceive(FMHeureReelle, &HeureReelle, sizeof(Horaire_t),
                WAIT_FOREVER);
                horaire=HeureReelle; // Peut etre faire champ a champ
                DivH10=0;
                char chaine[]="Corrigee";
                strcpy((Heure_Locale.provenance),chaine);
                Heure_Locale.horaire=horaire;
                msgQSend(FMHeureLocale, &Heure_Locale, sizeof(
                HeureLocale_t), WAIT_FOREVER, MSG_PRI_NORMAL);
        }
}


/*----------Affiche----------*/
void Affiche(void){
        //Declaration des variables locales
        HeureLocale_t Heure_Locale;
        //Procedure
        while(1){
                msgQReceive(FMHeureLocale, &Heure_Locale, sizeof(
                HeureLocale_t),WAIT_FOREVER);
                printf("Origine du message : %s\n",
                Heure_Locale.provenance);
                printf("l'heure est : %d heure, %d min, %d sec\n",
                Heure_Locale.horaire.heure,
                Heure_Locale.horaire.minute,
                Heure_Locale.horaire.seconde);
        }
}

/*----------Application----------*/
int start()
{
        //Creation des semaphores
        H10=semBCreate(SEM_Q_PRIORITY, SEM_EMPTY);
        FMHeureReelle=msgQCreate(1, sizeof(Horaire_t), MSG_Q_FIFO);
        FMHeureLocale=msgQCreate(1, sizeof(HeureLocale_t), MSG_Q_FIFO);


        //Demarrage des taches
```

```
        printf("Demarrage de l'estimation de l'horloge\n");
        tidEstime_Heure=taskSpawn("Estime_Heure", 13,0, STACK_SIZE,
        (FUNCPTR) Estime_Heure,0,0,0,0,0,0,0,0,0,0);

        printf("Demarrage de la correction de l'horloge\n");
        tidCorrige_Heure=taskSpawn("Corrige_Heure", 14,0, STACK_SIZE,
        (FUNCPTR)Corrige_Heure,0,0,0,0,0,0,0,0,0,0);

        printf("Demarrage de l'affichage\n");
        tidAffiche=taskSpawn("Affiche", 15,0, STACK_SIZE,
        (FUNCPTR)Affiche,0,0,0,0,0,0,0,0,0,0);

        startEnvironment(H10, FMHeureReelle);

        printf("Fin de start\n");
        return (EXIT_SUCCESS);
}

/*----------------Fermeture----------------*/
int stop()
{
        //printf("Debut de stop\n");
        stopEnvironment();

        //Destruction des taches
        taskDelete(tidEstime_Heure);
        taskDelete(tidCorrige_Heure);
        taskDelete(tidAffiche);

        //Desctruction des semaphores
        semDelete(H10);
        msgQDelete(FMHeureReelle);
        msgQDelete(FMHeureLocale);

        printf("Fin de stop\n");
        return(EXIT_SUCCESS);
}
```