Laure INTURRISI
5TS1

FINAL ASSIGNEMENT

Partie 1

Pour cette partie, le code met beaucoup de temps à tourner en entier. Cela vient évidemment du fait que le code est long à tourner dans tous les cas. En revanche en ajoutant des fonctions *print()* un peu partout afin de pouvoir suivre l'avancée de mon code fait que celui-ci est encore plus long.

Je n'ai pas réussi à faire exécuter le code en entier. Le code passait encore parmi toutes les permutations commençant par la tâche 11 alors qu'il tournait déjà depuis 12h. j'ai donc essayé de le faire tourner pour seulement 5 tâches afin de pouvoir me rendre compte ou non de l'efficacité du code lorsque τ_5 peux manquer une deadline. Cette fois-ci, le code a prit plus de 48h à seulement tester toutes les possibilités avec la tâche 11 étant la première tâche effectuée. Je pense que mon code n'est clairement pas optimal. Il se peut aussi que j'ai fait une erreur. Néanmoins le code fonctionne jusqu'à 4 tâches et une hyperpériode de 40, à partir de là, le temps de calcul est beaucoup trop long.

Pour avoir une vision plus complète de l'efficacité du code, voici ce qu'il me renvoie pour 3 tâches et une hyperpériode de 20 :

```
Meilleur scheduler :
Ordre: [11, 31, 21, 12, 22] | Temps d'attente minimal: 8

Aucun scheduler valide pour tau5 permettant de manquer une deadline.
```

Le code s'est exécuté instantanément, de même lorsque l'on choisit de le faire pour 4 tâches et une hyperpériode de 20. Or, dès que l'on passe à une hyperpériode de 40, le code est extrêmement plus long. Il passe de quelques secondes à 3/4h d'exécution.

Voici toutes les possibilités de schedule (les possibilités avec et sans la possibilité de manquer une deadline pour τ_5 étant les mêmes puisqu'il n'y a pas de tâche 5, avec beaucoup de tâches en moins par souci de longueur) :

```
Schedulers valides avec temps d'attente total :
Ordre: [11, 21, 12, 22, 31, 13, 23, 14, 24, 32] | Temps d'attente: 38
Ordre: [11, 21, 12, 22, 31, 13, 23, 14, 32, 24] | Temps d'attente: 37
Ordre: [11, 21, 12, 22, 31, 13, 23, 24, 14, 32] | Temps d'attente: 39
Ordre: [11, 21, 12, 22, 31, 13, 23, 24, 32, 14] | Temps d'attente: 39
Ordre: [11, 21, 12, 22, 31, 13, 23, 32, 14, 24] | Temps d'attente: 28
Ordre: [11, 21, 12, 22, 31, 13, 23, 32, 24, 14] | Temps d'attente: 29
Ordre: [11, 21, 12, 22, 31, 13, 32, 23, 14, 24] | Temps d'attente: 27
Ordre: [11, 21, 12, 22, 31, 13, 32, 23, 24, 14] | Temps d'attente: 28
Ordre: [11, 21, 12, 22, 31, 23, 13, 14, 24, 32] | Temps d'attente: 39
Ordre: [11, 21, 12, 22, 31, 23, 13, 14, 32, 24] | Temps d'attente: 38
Ordre: [11, 21, 12, 22, 31, 23, 13, 24, 14, 32] | Temps d'attente: 40
Ordre: [11, 21, 12, 22, 31, 23, 13, 24, 32, 14] | Temps d'attente: 40
Ordre: [11, 21, 12, 22, 31, 23, 13, 32, 14, 24] | Temps d'attente: 29
Ordre: [11, 21, 12, 22, 31, 23, 13, 32, 24, 14] | Temps d'attente: 30
Ordre: [11, 21, 12, 22, 31, 23, 32, 13, 14, 24] | Temps d'attente: 29
Ordre: [11, 21, 12, 22, 31, 23, 32, 13, 24, 14] | Temps d'attente: 30
[...]
Ordre: [11, 21, 12, 31, 22, 23, 32, 13, 24, 14] | Temps d'attente: 29
Ordre: [31, 21, 11, 12, 22, 32, 23, 13, 24, 14] | Temps d'attente: 19
Ordre: [31, 21, 11, 22, 12, 13, 23, 14, 24, 32] | Temps d'attente: 29
Ordre: [31, 21, 11, 22, 12, 13, 23, 14, 32, 24] | Temps d'attente: 28
Ordre: [31, 21, 11, 22, 12, 13, 23, 24, 14, 32] | Temps d'attente: 30
Ordre: [31, 21, 11, 22, 12, 13, 23, 24, 32, 14] | Temps d'attente: 30
Ordre: [31, 21, 11, 22, 12, 13, 23, 32, 14, 24] | Temps d'attente: 19
[...]
Ordre: [31, 21, 11, 22, 12, 23, 32, 13, 14, 24] | Temps d'attente: 20
Ordre: [31, 21, 11, 22, 12, 23, 32, 13, 24, 14] | Temps d'attente: 21
Ordre: [31, 21, 11, 22, 12, 32, 13, 23, 14, 24] | Temps d'attente: 18
Ordre: [31, 21, 11, 22, 12, 32, 13, 23, 24, 14] | Temps d'attente: 19
Ordre: [31, 21, 11, 22, 12, 32, 23, 13, 14, 24] | Temps d'attente: 19
Ordre: [31, 21, 11, 22, 12, 32, 23, 13, 24, 14] | Temps d'attente: 20
```

Partie 2

Pour construire le fixed priority scheduler, il faut tout d'abord fixer les priorités comme tel :

```
#define task1priority                    ( tskIDLE_PRIORITY + 5 )
#define task2priority                    ( tskIDLE_PRIORITY + 4 )
#define task3priority                    ( tskIDLE_PRIORITY + 3 )
#define task4priority                    ( tskIDLE_PRIORITY + 2 )
#define task5priority                    ( tskIDLE_PRIORITY + 1 )
```

Ensuite, il faut définir la vitesse à laquelle les données sont envoyées et reçues dans la file d'attente. Cela représente le délai entre les exécutions successives de la tâche :

```
#define mainTASK_SEND_FREQUENCY_MS          pdMS_TO_TICKS( 200UL )
#define mainTIMER_SEND_FREQUENCY_MS         pdMS_TO_TICKS( 2000UL )

#define mainVALUE_SENT_FROM_TASK            ( 100UL )
#define mainVALUE_SENT_FROM_TIMER           ( 200UL )
```

Puis, il faut définir le de type de fonction des tâches de la manière suivante :

```
static void task1(void * pvParameters);
static void task2(void * pvParameters);
static void task3(void * pvParameters);
static void task4(void * pvParameters);
static void task5(void * pvParameters);
```

Alors, il est enfin possible de pouvoir créer les tâches :

```
xTaskCreate(task1, "Rx", configMINIMAL_STACK_SIZE, NULL, task1priority,
NULL );
xTaskCreate(task2, "TX", configMINIMAL_STACK_SIZE, NULL, task2priority,
NULL );
xTaskCreate(task3, "FX", configMINIMAL_STACK_SIZE, NULL, task3priority,
NULL );
xTaskCreate(task4, "ZX", configMINIMAL_STACK_SIZE, NULL, task4priority,
NULL );
xTaskCreate(task5, "GX", configMINIMAL_STACK_SIZE, NULL, task4priority,
NULL );
```

Pour réaliser le FreeRTOS task scheduler on utilise à présent la fonction *vTaskStartScheduler()*. Finalement, j'utilise une boucle infinie pour faire tourner le code jusqu'à ce que l'utilisateur décide de l'arrêter

Pour comprendre comment les tâches sont créées, prenons l'exemple de la tâche 1. Son but était d'imprimer « Working ». J'ai commencé par définir trois variables locales *xNextWakeTime*, *xBlockTime* et *ulValueToSend*. La première est utilisée pour stocker l'heure à laquelle la prochaine exécution de la tâche doit avoir lieu. La deuxième représente le délai entre les exécutions successives de la tâche. Enfin, le troisième n'est pas utilisé ici, mais peut être une valeur à envoyer à une autre tâche. Pour initialiser *xNextWakeTime*, j'ai utilisé la fonction *xTaskGetTickCount()* pour lui permettre de stocker le nombre de ticks écoulés depuis le

démarrage du système. Cette variable sera ensuite utilisée pour planifier les futures exécutions de la tâche.

Ensuite, j'initialise une boucle infinie pour que la tâche puisse effectuer son action. « Working » s'affiche, puis la fonction *vTaskDelayUntil()* est appelée pour mettre la tâche en attente. Ce temps d'attente est défini par *xBlockTime* à partir du temps défini par *xNextWakeTime*. De cette manière, les futures exécutions de la tâche peuvent être programmées avec un délai constant.

Les quatre autres tâches fonctionnent de la même manière, à l'exception de leurs actions bien sûr. La tâche 2 convertit une température de Fahrenheit en Celsius. La tâche 3 multiplie deux entiers longs fixes et affiche le résultat. La troisième tâche effectue une recherche binaire dans une liste fixe de 50 éléments. Finalement la dernière, elle vérifie à intervalle régulier, toutes les 200 ms, si l'utilisateur presse la touche 1, alors RESET est activé. Elle affiche alors son nouvel état avec : « RESET flag received: 1 » puis réinitialise son état.