Tanguy DEREN - 5TS1

PROJET :

Systèmes d'exploitation temps-réel pour l'embarqué

PARTIE 1

Pour cette section, l'exécution du code prend énormément de temps. Cela est principalement dû à la longueur intrinsèque du traitement. L'ajout de la fonction print() à chaque permutation pour suivre l'avancement du programme a considérablement ralenti le processus.

Le code n'a jamais tourné jusqu'à la fin car ça prenait beaucoup trop de temps. En effet, le programme continuait à analyser toutes les permutations commençant par la tâche 11, même après 12 heures d'exécution. J'ai alors limité le nombre de tâches à cinq afin de tester l'efficacité du code lorsque $\tau 5$ peut manquer une deadline. Cette fois, le code a mis plus de 48 heures pour simplement tester les configurations où la tâche 11 était exécutée en premier. Le programme fonctionne correctement jusqu'à 4 tâches et une hyperpériode de 40. Au-delà de cela, le temps de calcul devient beaucoup trop élevé.

Voici ce que retourne le code avec 3 taches et une hyperpériode de 20 :

```
Planification optimale :
Ordre des jobs : [11, 31, 21, 12, 22] | Temps d'attente minimal : 8

Aucune planification valide pour tau5 avec une deadline manquée autorisée.
```

PARTIE 2

Pour construire le programme de planification il fait fixer l'ordre de priorité des tâches comme cela :

```
/* Priorities at which the tasks are created. */
#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 transmises et reçues dans la file d'attente. Cela représente le délai entre les exécutions des tâches.

```
/* The rate at which data is sent to the queue.  The times are converted from
 * milliseconds to ticks using the pdMS_TO_TICKS() macro. */
#define mainTASK_SEND_FREQUENCY_MS          pdMS_TO_TICKS( 200UL )
#define mainTIMER_SEND_FREQUENCY_MS         pdMS_TO_TICKS( 2000UL )

/* The values sent to the queue receive task from the queue send task and the
 * queue send software timer respectively. */
#define mainVALUE_SENT_FROM_TASK            ( 100UL )
#define mainVALUE_SENT_FROM_TIMER           ( 200UL )
```

On vient par la suite définir le type de fonction des tâches :

```
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);
```

Enfin, on 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 comprendre le processus de création des tâches, prenons l'exemple de la première tâche, qui a pour objectif d'afficher "Working". Pour commencer, j'ai commencé par définir trois variables locales : xNextWakeTime, xBlockTime et ulValueToSend. La première variable est utilisée pour mémoriser l'instant où la tâche doit se relancer. La deuxième est un délai, qui détermine le temps d'attente entre chaque exécution de la tâche. Quant à la troisième, elle n'est pas exploitée dans cet exemple, mais elle pourrait représenter une donnée destinée à une autre tâche.

Pour initialiser xNextWakeTime, j'ai fait appel à la fonction xTaskGetTickCount(), qui permet de récupérer le nombre de ticks écoulés depuis le démarrage du système. Cette valeur servira à programmer les prochaines exécutions de la tâche.

Ensuite, j'ai créé une boucle infinie pour permettre à la tâche d'effectuer son action en continu. À chaque cycle, le message "Working" est affiché, puis la fonction vTaskDelayUntil() est

utilisée pour suspendre l'exécution de la tâche pendant un délai déterminé par xBlockTime, en fonction de xNextWakeTime. Ce mécanisme garantit que les futures exécutions se produiront avec une périodicité constante.

Le processus est similaire pour les quatre autres taches à l’exception bien sûr de leurs actions respectives.