TP1 : OBSERVATION du comportement de FREE RTOS

Debut de création d'un OS maison imitant FREE RTOS

Le but de ce TP est de comprendre les concepts d'un OS temps réel en réalisant un micro OS multi-tâches capable de faire de la commutation de contexte, et de mettre des tâches en attente pour une durée donnée. Cela donne la démarche de vérification quand on porte FREERTOS sur une cible nouvelle, et indique une démarche de test et vérification lors de la mise en place.

Bien suivre l'ordre de réalisation conseillé et ne pas chercher à aller trop vite :

1 Simuler le projet Free RTOS donné :

Ce projet permet de simuler la commutation entre 4 tâches simple qui font clignoter des Leds

1.1 Simuler le projet de base , repérer les trois interruptions clefs de l'OS :

pour les trois interrutions suivantes, indiquer le nom de la fonction, le nom du fichier, la ligne

- **Interruption permettant de lancer la première tâche pour le démarrage de l'OS**
- **Interrution permettant de gérer le temps dans l'OS (timer Systick)**
- **Interruption dédiée à la commutation de tâche dans l'OS**

remplir le tableau suivant :

informations	avant le démarrage de l'OS	dans la première tâche	dans l'IT gérant le temps dans l'OS	dans la tâche suivante
MODE Thread/Handler				
STACK (MSP/PSP)				
R13(SP)				
R14				

1.2 forcer une commutation de tâche

Editer dans le fichier FREERTOSConfig.h le #define configTICK_RATE_HZ pour descendre à seulement 1000 commutations de tâches par seconde.

Constater qu'on ne commute plus souvent du tout en simulation.

En utilisant la fonction taskYIELD() de l'API de FREE RTOS déclencher des commutations de tâches volontaires

Afficher les chronogrammes des pattes P3.25 P3.26 P0.22 ainsi que ceux de P2.4 P2.5 P2.6 et P2.7

Regarder l'étape d'une création de tâche dans l'OS, regarder la fabrication de la fausse pile.

1.3 Réalisation de l'exercice du cours 1 avec 4 tâches qui gèrent 3 clignoteurs et un bouton

On cherchera dans l'API ou dans le résumé donné comment implémenter un délai en FREE RTOS

implémenter les 3 tâches des clignoteurs

implémenter la scruptation du bouton pour changer les paramètres de la tâche 3

Regarder dans le simulateur comment est géré un délai, est il bloquant ?

afficher les chronogrammes.

2 lire le texte annexe donné et déclarer les variables:

- **Dans le projet FREERTOS, repérer les fichiers Port.c , Portmacro.h où copier des morceaux de code pour notre propre projet : repérer les noms des fonctions et IT utiles :**
  - It liée à la commutation de tâche :
  - It liée au lancement de la première commutation :
  - It gestion du temps :
  - fonction d'initialisation de l'OS :
  - fonction de choix de la prochaine tâche :
  - fonction permettant la création d'une tâche :
  - fonction d'initialisation d'une fausse pile pour une tâche :
  - fonction armant une commutation de tâche :
- **Pour commencer, commenter les déclarations incomplètes pour vérifier que le projet compile et que votre chaîne de developpement est fonctionnelle.**
- **Relire l'annexe du texte du TP et définir la structure du TCB, et les variables de l'OS**
  On pourra s'inspirer du résumé de freeRTOS et de l'annexe.
- **Déclarer les tableaux nécessaires au fonctionnement de notre OS**
  (TABLEAU de tous les TCB gérés par notre OS et PILES des tâches)

Ouvrir le fichier map et repérer où sont rangées les diverses variables de L'OS :

On ouvrira systématiquement un onglet WATCH pour les structures des TCB et les variables de l'OS, et des onglets memory pour les piles pour espionner l'état des variables.

Il faudra faire cela le plus régulièrement possible, bien lire les étapes qui suivent avant d'improviser.

3 Ecrire la fonction Task_create :

- Reprendre le cours pour voir tous les paramètres nécessaires à l'appel de la fonction Task_create
- S'inspirer dans le manuel de la fonction xTaskCreateStatic() page 39 du Référence Manuel

être capable de préparer un TCB et la pile associée

dans creation_des_taches(), on ne crée aucune tâche, seul la tâche idle existera donc...

à condition qu'on n'oublie pas de la créer dans Lancement_OS()...

On peut tester le bon déroulement du code, et en espionnant le tableau de la pile choisie regarder si tout se passe bien en confrontant aux informations du fichier MAP, même si Lancement_OS() est incomplet, et ne contient que l'appel à la création de la tâche IDLE. L'OS n'est pas encore lancé pour l'instant.

Si la pile semble correctement constituée, alors on peut tenter de passer à l'étape d'après.

On complète alors petit à petit Lancement_OS() lors des prochaînes étapes en s'inspirant du code de FREE RTOS

4 Ecriture de Lancement_OS(), sans configuration du Systick, juste initialiser les variables de l'OS, configurer juste l'IT SVC_Handler et le lancement de la première tâche.

4.1 Dans un premier temps, on est pas obligé de remplir SVC_Handler avec un code complexe, on se contente de vérifier qu'on va l'atteindre. On s'affranchira de replacer la pile MSP à son origine.

Si la fonction en assembleur ne se lance pas à l'execution de l'instruction SVC:

- Vérifier si on part bien en interruption, en executant en pas à pas.
- si le vecteur d'interruption est mal configuré, on ne lance pas la bonne fonction.

4.2 Compléter le code de SVC_Handler, et reprendre la simulation.

On pourra ne pas recopier la réinitialisation de la pile MAIN.

Vérifier que les registres sont correctement bien initialisés par l'instruction de dépilement depuis R0

Vérifier que la fin du dépilement s'exécute bien et qu'on passe bien la main à la seule tâche (idle)

En cas de soucis, vérifier juste après plantage l'état des registres pour comprendre pourquoi le registre R15(PC) n'a pas été initialisé correctement.

Dans la Tâche idle, on ne met aucun code provisoirement (juste les commande des leds), vérifier qu'on se retrouve bien dans la bonne fonction en sortant de la tâche, expliquer pourquoi.

Que se passe t il si on commente la ligne orr r14, #0xd de l'exception SVC_Handler ?