TP MI03 - Réalisation d'un mini noyau temps réel ARM (Parties 1 et 2)

Introduction

Ce TP consiste à réaliser, sur une carte ARM ARMADEUS et en langage C un mini noyau temps réel très simplifié. Le but de ce noyau est de pouvoir répondre correctement au programme test proposé dans le fichier *NOYAUTES.C*

Les services proposés à l'utilisateur dans un premier temps sont les suivants :
- commencer une première tâche : *start(tâche)*
- créer une tâche : *cree(tâche)*
- activer une tâche : *active(tâche)*
- appel au scheduler : *scheduler()*

Ces services sont ceux qui apparaissent dans le programme test ou qui seront utilisés par la suite. Dans un deuxième temps, on ajoutera des services de synchronisation et de communication inter-tâches.

Le noyau proposé est préemptif, c'est-à-dire que les appels au noyau peuvent être explicites dans les tâches ou les primitives du noyau, ou provoqués par le noyau temps réel. Il peut donc y avoir des appel au service d'interruption dues au système extérieur ou à l'horloge interne.

A chaque tâche est associé un contexte. Tous les contextes sont rangés dans un tableau. On ne peut avoir que huit tâches au maximum.

Le contexte associé à chaque tâche est constitué de :
- une adresse de début de tâche,
- l'état de la tâche qui peut être :
  - non crée *NCREE*
  - créé ou dormant *CREE*
  - prêt *PRET*
  - en exécution *EXEC*
- un pointeur de pile initial,
- un pointeur de pile courant en mode *IRQ*.

Une tâche est décrite comme un sous-programme (fonction C) de type *TACHE*.

Tous les types et constantes, et prototypes de fonctions utilisés dans le noyau sont définis dans le fichier **NOYAU.H** ; il est donc recommandé de consulter de façon approfondie ce fichier.

Les deux fichiers principaux sont :

- *NOYAU.C* noyau
- *NOYAUFIL.C* procédures de gestion de la file de tâches

Les fichiers *imx_*.** contiennent du code permettant d'accéder aux périphériques utiles pour le TP.
Les fichiers *serialio.** permettent des entrées et sortie sur le port série de la carte. On peut visualiser ces sorties par l'intermédiaire d'un terminal série adéquate.

1ère partie : Ordonnanceur de tâches

Son rôle est de gérer l'activité des tâches prêtes ou en exécution. Toutes les tâches ont la même priorité. La gestion de leur activité se fait en FIFO, c'est-à-dire que la plus ancienne dans la file sera la première activée.

La solution choisie dans ce noyau minimum consiste à mémoriser les numéros des tâches dans un tableau *_file* de dimension *MAX_TACHE*. L'indice d'un élément du tableau correspond à un numéro de tâche et l'élément du tableau à la tâche suivante. La variable *_queue* contient l'élément qui pointe sur la prochaine tâche à activer.
Dès qu'une tâche devient active, elle est automatiquement remise en fin de file, puisqu'elle sera la dernière à être réactivée.

Les fonctions qui gèrent cette file doivent être écrites dans le fichier *NOYAUFIL.C* et sont les suivantes :

- *file_init()* : initialise la file. *_queue* contient une valeur de tâche impossible, *F_VIDE*, indiquant ainsi que la file est vide.
- *ajoute(n)* : ajoute la tâche n en fin de file, elle sera la dernière à être activée
- *suivant()* : retourne la tâche à activer, et met à jour *_queue* pour qu'elle pointe sur la suivante.
- *retire(n)* : retire la tâche *n* de la file sans en modifier l'ordre.

Exemple

On suppose la situation initiale suivante :

Début				Fin
3	5	1	0	2

Tâche active 2

Le tableau est donc :

			_queue					
Indice	0	1	2	3	4	5	6	7
_file	2	0	3	5		1		

suivant()

Début				Fin
5	1	0	2	3

Tâche active 3

Le tableau est donc :

			_queue					
Indice	0	1	2	3	4	5	6	7
_file	2	0	3	5		1		

retire(0)

Début			Fin				
5	1	2	3				

Tâche active 3

Le tableau est donc :

				_queue			
Indice	0	1	2	3	4	5	6	7
_file		2	3	5		1		

ajoute(6)

Début				Fin			
5	1	2	3	6			

Tâche active 3

Le tableau est donc :

							_queue	
Indice	0	1	2	3	4	5	6	7
_file		2	3	6		1	5	

Questions

Ecrire les fonctions décrites ci-dessus en complétant le fichier *NOYAUFIL.C.*
Ecrire un petit programme *TESTFILE.C* qui permet de tester les différentes fonctions.

2ème partie : gestion et commutation de tâches

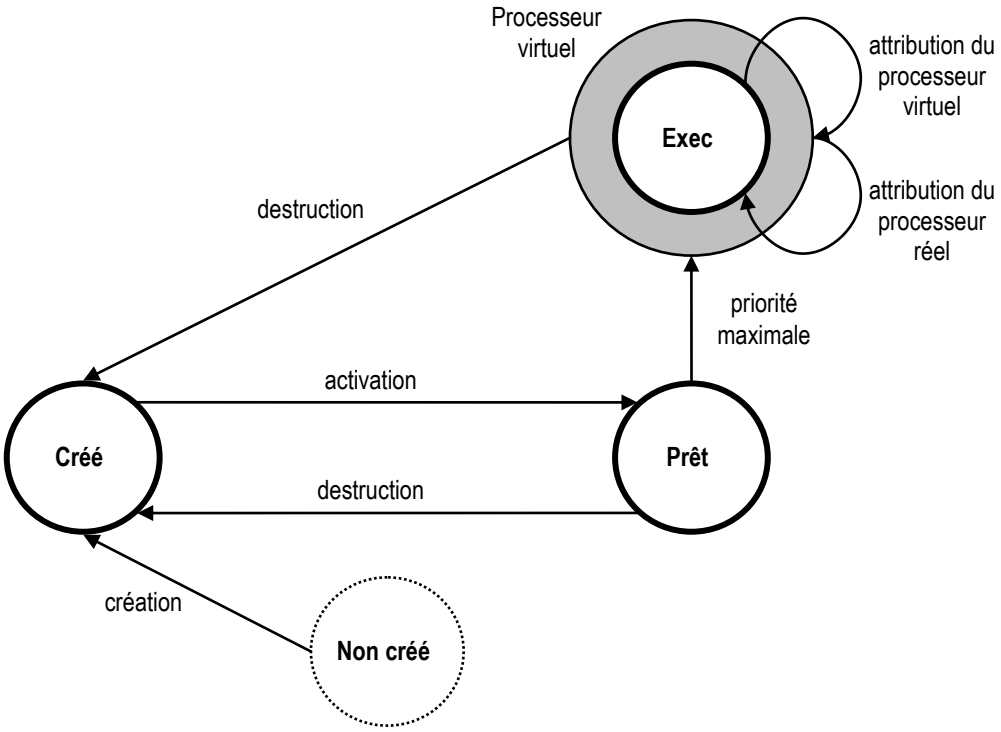
Cette partie du TP consiste à réaliser les primitives de gestion des tâches du mini noyau temps réel, ainsi que le système de commutation de tâches.

Etat des tâches

Les tâches peuvent prendre l'un des quatre états suivants :

- **Non créée (NCREE) :** la tâche n'existe pas, elle n'est pas connue du noyau.
- **Créée (CREE) :** la tâche est connue du noyau, une pile lui a été allouée ainsi qu'un identifiant, elle est prête à être activée. Elle reste dans cet état tant qu'un événement ne la fait pas évoluer.
- **Prête (PRET) :** une tache à l'état prêt est candidate au processeur. Son lancement ne dépend que de sa priorité par rapport aux tâches en cours d'exécution ou aux autres tâches à l'état prêt. C'est l'ordonnanceur qui décide.
- **En exécution (EXEC) :** une tache en exécution est une tache en possession du processeur virtuel ou du processeur réel. Dans un système monoprocesseur, plusieurs tâches de même

priorité peuvent être en exécution mais une seule tâche est en possession à un instant donné du processeur réel. C'est l'ordonnanceur qui décide laquelle.



Description du contexte d'une tâche

Chaque tache est associée à un contexte. Tous les contextes sont rangés dans le tableau *_contexte*. On ne peut avoir que 8 tâches au maximum.
Le contexte associé à la tâche est constitué de :

- Une adresse de début de la tache *tache_adr*
- L'état de la tache *status*
- Un pointeur de pile initial *sp_ini*
- un pointeur de pile courant en mode *IRQ* : sp_irq

L'état du processeur (registres, pointeur d'instruction) est doit être sauvegardé dans la pile de la tâche courante lors de l'appel au scheduler, que cet appel soit dû à l'horloge interne ou à l'application. La sauvegarde et la restitution du contexte dans le scheduler se fait donc en manipulant le pointeur de pile courant du mode d'interruption du processeur.

Services proposés

- démarrer le noyau et commencer une première tâche : *start(adr_tache)*
    - initialise les structures de données du noyau, met en place le gestionnaire d'interruption *scheduler*
    - crée et active la première tâche, dont l'adresse est passée en paramètres
- créer une nouvelle tâche : *cree(adr_tache)*
    - alloue un espace dans la pile à la tâche et lui attribue un identifiant, qui est retourné à l'appelant.
- activer une tâche : *active(tache)*
    - place la tache dans la file des tâches prêtes

- détruire une tâche : *fin_tache()*
    - change l'état d'une tâche active en CREE. La tâche peut alors être relancée à partir du début. Elle est retirée de la file des tâches prêtes.
- appel au gestionnaire de tâches (scheduler) : *schedule()*.
- sortie du noyau : *noyau_exit()*.

Chaque tâche est une fonction C, de type *TACHE*. Dans les primitives décrites ci-dessus, *adr_tache* représente l'adresse de la fonction associée à une tâche, et *tache* le numéro de la tâche.

Variables globales du noyau

- *_tos (top of stack)* : point de la pile à partir duquel il est possible d'allouer de l'espace pour une nouvelle tâche
- *_tache_c* : identifiant de la tâche courante, en exécution sur le processeur physique.
- _contexte[MAX_TACHES] : tableau de contextes des tâches
- compteurs[MAX_TACHES] : tableau permettant de comptabiliser individuellement le nombre d'activation par tâche.
- _ack_timer : drapeau permettant de différencier une activation du service d'interruption par le Timer ou le logiciel.

Questions

Ecrire les fonctions décrites ci-dessus en complétant le fichier *NOYAU.C*.
Tester le noyau au moyen du programme *NOYAUTES.C*.