



Institut  
Mines-Télécom

# FreeRTOS

ELECINF344

Samuel TARDIEU <sam@rfc1149.net>  
Février 2013



## FreeRTOS : configuration

La configuration de FreeRTOS se fait dans un fichier `FreeRTOSConfig.h`, et définit notamment :

- les fonctionnalités utilisées (préemption, sémaphores, files d'attente, priorités dynamiques ou non, délais relatifs, délais absolus, etc.) ;
- le nombre de niveau de priorités utilisées, et la plage réservée à l'exécutif de FreeRTOS et aux interruptions ;
- la taille de pile minimale pour chaque tâche, la taille de la mémoire disponible pour le tas ;
- la fréquence du *tick* système, qui représente la granularité des délais offerts au programmeur et la fréquence du changement de tâche active entre tâches de même priorité.

# FreeRTOS : synchronisation

FreeRTOS offre les outils suivants pour la synchronisation entre les tâches et avec les routines d'interruptions :

- files d'attente ;
- sémaphores binaires ou à valeur maximale fixée, sans héritage de priorité ;
- verrous avec héritage de priorité.

Tous ces outils peuvent être utilisés depuis des tâches avec des *timeouts*, ou depuis des interruptions avec des primitives dédiées non bloquantes.

# FreeRTOS : concurrence

Pour la gestion de programmes concurrents, FreeRTOS propose :

- un système de tâches préemptif avec gestion de la priorité ;
- un système de coroutines avec gestion de la priorité ;
- un système de *hooks* permettant de spécifier du code exécuté à chaque *tick* du noyau ou lorsque le système exécute l'*idle-task*.

Ces systèmes peuvent être utilisés seuls ou combinés. Il est par exemple possible d'utiliser des coroutines lors de l'exécution de l'*idle-task* et des tâches en mode préemptif sinon.

# FreeRTOS : gestion du temps

Le système de *ticks* de FreeRTOS permet :

- à une tâche d'être suspendue pendant un nombre entier de ticks ;
- à une tâche d'être suspendue jusqu'à une date donnée (en nombre de *ticks* depuis le démarrage) ;
- à l'ordonnanceur de changer de tâche active parmi les tâches de plus haute priorité lors du déclenchement du *tick*.

Plus la fréquence du *tick* est grande, plus la gestion du temps est précise, mais plus le surcoût lié à cette gestion est important.

Pour des événements plus précis, il est possible d'utiliser des *timers* externes à FreeRTOS qui débloquent des tâches à l'aide d'un sémaphore binaire.

# FreeRTOS : gestion de la mémoire

FreeRTOS a besoin de routines `malloc()` et `free()` pour allouer la mémoire des différentes entités. En addition, trois implémentations sont livrées avec FreeRTOS :

- `heap_1.c` : allocation possible, désallocation ignorée ;
- `heap_2.c` : allocation possible, désallocation possible sans agrégation des zones libres contiguës ;
- `heap_3.c` : utilise les fonctions `malloc()` et `free()` fournies, mais les rend *thread-safe*.

Si les tâches et les objets (sémaphores, files d'attentes, etc.) sont créés au début du programme, toutes les allocations disponibles seront faites avant de démarrer l'ordonnanceur. Il sera par la suite impossible de manquer de mémoire.

## FreeRTOS : outils de trace

FreeRTOS est doté d'outils de trace et de *debug* permettant de suivre le fonctionnement du système et de s'assurer de son bon déroulement :

- les tâches peuvent être nommées et la liste des tâches consultées ;
- des données supplémentaires peuvent être attachées aux tâches, et consultées lors du changement de contexte (par exemple en changeant une sortie analogique pour représenter une tension différente pour chaque tâche) ;
- un certain nombre d'attributs peuvent être consultés ;
- des fonctionnalités de vérification de non-débordement de la pile peuvent être employés (vérification lors du changement de contexte et canari).

# FreeRTOS : anatomie d'une application

Une application FreeRTOS aura généralement la forme suivante :

1. Initialisation sommaire du matériel (assez pour les deux étapes suivantes)
2. Copie de la section `.data` et initialisation de la section `.bss`
3. Exécution du programme principal (à partir de l'étape suivante)
4. Fin de l'initialisation du matériel (horloges supplémentaires, *wait states*, etc.)
5. Initialisation des périphériques
6. Création des structures FreeRTOS (tâches et structures de contrôle)
7. Démarrage de l'ordonnanceur