FreeRTOS

SE302

Samuel TARDIEU <sam@rfc1149.net>

Septembre 2020

FreeRTOS

- Système d'exploitation temps-réel pour microcontrôleurs ;
- Logiciel libre, licence MIT depuis l'acquisition par Amazon ;
- Multi-tâches, préemptif, avec coroutines (rarement utilisées) ;
- Très petit, très rapide, très bien documenté ;
- S'interface facilement avec AWS sous le nom de *Amazon FreeRTOS*.

FreeRTOS : configuration

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

- les fonctionalités 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 : nommage

Les variables et fonctions de FreeRTOS respectent une convention de nommage :

- un préfixe en minuscules (par exemple `ul` pour *unsigned long*, `v` pour *void*, `x` pour une structure) détermine le type ;
- les fonctions de l'*API* référencent le fichier qui les définit, ou `prv` pour *private* : `vTaskDelete()` est défini dans `tasks.c` et retourne `void` ;
- les macros sont préfixées par une référence au fichier où elles sont définies : `configUSE_PREEMPTION` vient de `FreeRTOSConfig.h`.

La documentation de FreeRTOS décrit plus précisément cette convention.

FreeRTOS : synchronisation

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

- files d'attente ;
- flux d'octets ou de messages (écrivain unique, lecteur unique) ;
- sémaphores binaires ou à valeur maximale fixée ;
- verrous avec héritage de priorité ;
- notifications de tâches (simples ou multiples depuis 10.4).

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 débloqueront des tâches à l'aide d'un sémaphore binaire.

FreeRTOS : *tickless* mode

FreeRTOS peut :

- suspendre les ticks systèmes lorsque le système est inactif (*idle*) si le prochain événement de scheduling est placé suffisamment loin dans le temps ;
- corriger automatiquement le compteur de ticks système lors du réveil afin de prendre en compte la période de sommeil passée (mode *builtin*) ;
- appeler des fonctions de l'utilisateur avant et après l'endormissement pour corriger l'horloge.

Cela permet au système de rester en veille pendant de longues périodes sans se réveiller périodiquement.

FreeRTOS : gestion de la mémoire (1/2)

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

- `heap_1.c` : allocation possible, désallocation ignorée (obsolète, l'allocation statique est disponible) ;
- `heap_2.c` : allocation possible, désallocation possible sans agrégation des zones libres contiguës (obsolète, cf. `heap_4.c`) ;
- `heap_3.c` : utilise les fonctions `malloc()` et `free()` fournies, mais les rend *thread-safe* ;
- `heap_4.c` : *first fit* avec regroupement des blocs contigus ;
- `heap_5.c` : idem, mais le tas peut aussi être réparti entre plusieurs zones de mémoire non contigües.

FreeRTOS : gestion de la mémoire (2/2)

Dans tous les cas, il vaut mieux privilégier une allocation statique, afin de vérifier la disponibilité de toute la mémoire nécessaire dès l'édition de liens.

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 fonctionalités 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

FreeRTOS+

FreeRTOS+ est un ensemble de bibliothèques additionnelles fournies avec FreeRTOS :

FreeRTOS+TCP : sockets TCP ;

FreeRTOS+UDP : sockets UDP ;

FreeRTOS+CLI : interpréteur de commandes (*command line interpreter*) ;

FreeRTOS+IO : fournisseur d'abstractions de fonctions `open()`, `read()`, etc.

FreeRTOS : bibliothèques IoT

FreeRTOS+ vient également avec des bibliothèques supplémentaires dédiées aux objets connectés, dont :

MQTT : client pour un *broker* pub/sub MQTT ;

HTTPS : client https ;

OTA : mise à jour du firmware à distance.

Certaines fonctionnalités nécessitent l'utilisation d'Amazon AWS.