Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

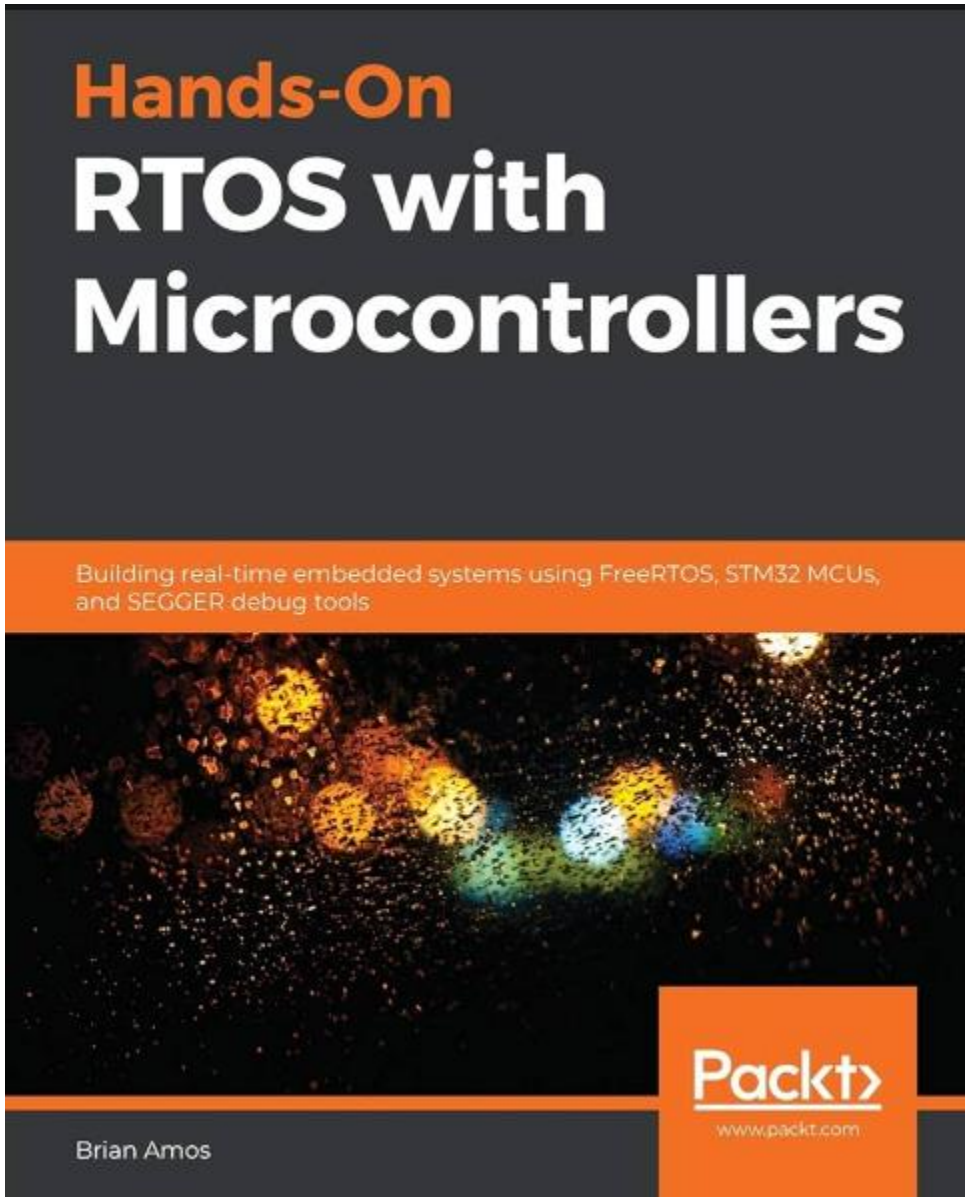
# CEG4566/CSI4541/SEG4545

# Conception de systèmes informatiques en temps réel

Hiver 2024

Professeur : Mohamed Ali Ibrahim, ing., Ph.D.

# Source:


Hands-On
RTOS with
Microcontrollers
Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools
Brian Amos
Packt
www.packt.com

# Chapitre 18 : Choisir une API RTOS

# Plan

- Comprendre les API génériques des RTOS
- Comparer FreeRTOS et CMSIS-RTOS
- Comparer FreeRTOS et POSIX
- Décider de l'API à utiliser

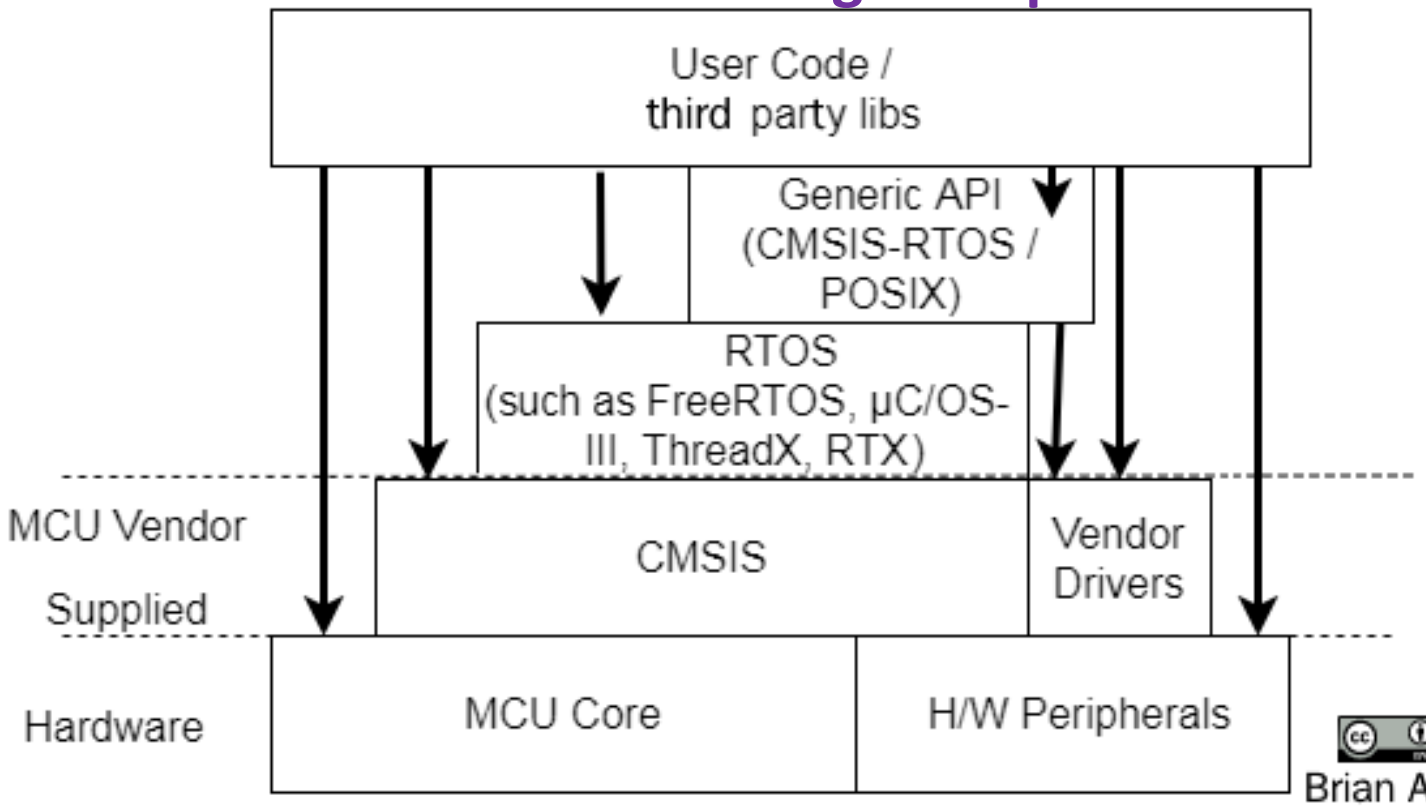
# Comprendre les API génériques du RTOS

- Une API RTOS définit l'interface de programmation avec laquelle l'utilisateur interagit lorsqu'il utilise le RTOS.
- Les API natives exposent toutes les fonctionnalités du RTOS.
- Jusqu'à présent, nous n'avons utilisé que l'API native de FreeRTOS.
- Cela a été fait pour faciliter la recherche d'aide pour une fonction donnée et pour exclure toute possibilité d'une couche d'enveloppe mal conçue entre FreeRTOS et une API générique.
- Cependant, ce n'est pas la seule option d'API pour FreeRTOS.
- Il existe également des API génériques qui peuvent être utilisées pour s'interfacer avec les fonctionnalités du RTOS
  - mais au lieu d'être liés à un RTOS spécifique, ils peuvent être utilisés sur plusieurs systèmes d'exploitation.

# Comprendre les API génériques du RTOS

- Ces API génériques sont généralement mises en œuvre sous la forme d'une couche enveloppante au-dessus de l'API RTOS native (à l'exception de RTX, qui ne dispose que de l'API CMSIS-RTOS).
- Nous pouvons voir ici où se trouve une API typique dans une pile générique de microprogrammes pour machines RISC avancées (**generic Advanced RISC Machines, ARM)** :

**Placement de l'API générique ARM**




Brian Amos

# Comprendre les API génériques du RTOS

- Comme le montrent les flèches du diagramme précédent, il n'existe pas d'abstraction unique qui empêche le code utilisateur d'accéder au niveau de fonctionnalité le plus bas.
- Chaque couche ajoute une autre API potentielle à utiliser, alors que la fonctionnalité de niveau inférieur reste disponible.
- Il existe deux API génériques qui peuvent être utilisées pour accéder à un sous-ensemble des fonctionnalités de FreeRTOS :
  - CMSIS-RTOS : ARM a défini une API indépendante du fournisseur pour les MCU, appelée Cortex Microcontroller Software Interface-RTOS (CMSIS-RTOS).
  - POSIX : L'interface de système d'exploitation portable (Portable Operating System Interface, POSIX) est un autre exemple d'interface générique commune à plusieurs fournisseurs et matériels. Cette API est plus couramment utilisée dans les systèmes d'exploitation à usage général, tels que Linux.

# Avantages des API génériques

- L'utilisation d'une API RTOS générique telle que CMSIS-RTOS ou POSIX présente plusieurs avantages pour les programmeurs et les fournisseurs d'intergiciels.
- Un programmeur peut écrire un code une seule fois et l'exécuter sur plusieurs MCU, en changeant de RTOS à volonté avec peu ou pas de changements dans le code de son application.
- Les fournisseurs d'intergiciels sont également en mesure d'écrire leur code pour interagir avec une API unique et prendre ensuite en charge plusieurs RTOS et matériels.

# Avantages des API génériques

- Comme vous l'avez peut-être remarqué dans le diagramme précédent, les API CMSIS-RTOS et POSIX ne nécessitent pas un accès exclusif à FreeRTOS.
- Puisque ces API sont implémentées comme des couches au-dessus de l'API native de FreeRTOS, le code peut utiliser soit l'API plus générique, soit l'API native de RTOS en même temps.
- Ainsi, il est parfaitement acceptable que certaines parties d'une application utilisent l'interface CMSIS-RTOS alors que d'autres utilisent l'API native de FreeRTOS.

# Avantages des API génériques

- Par exemple, si un fournisseur d'interface graphique livre son code et qu'il s'interface avec CMSIS-RTOS, rien n'empêche un développement supplémentaire avec l'API native de FreeRTOS.
- Le code du fournisseur d'interface graphique peut être intégré en utilisant CMSIS-RTOS, tandis que d'autres codes du système utilisent l'API FreeRTOS native, sans l'enveloppe CMSIS-RTOS.

# Désavantages des API génériques

- Ce qu'une API à usage général gagne en uniformité, elle le perd en spécificité.
- Une implémentation à usage général et à taille unique doit être suffisamment générique pour être applicable à la majorité des RTOS.
- Cela conduit à ce que les parties uniques soient exclues de l'interface standardisée, qui peut parfois inclure des caractéristiques très intéressantes.

# Désavantages des API génériques

- Étant donné que les vendeurs de RTOS ne sont pas toujours ceux qui assurent le support de CMSIS-RTOS, il est possible que la version de CMSIS-RTOS livrée soit en retard par rapport au cycle de publication du RTOS.
- Cela signifie que les mises à jour RTOS pour CMSIS-RTOS peuvent ne pas être incluses aussi souvent que pour l'API native.

# Désavantages des API génériques

- Il y a aussi le problème de l'obtention d'une aide en cas de problème
  - un fournisseur de RTOS sera généralement plus enclin (et capable) d'aider avec le code qu'il a réellement fourni.
- Souvent, il sera très difficile d'obtenir la prise en charge d'une abstraction que le fournisseur du RTOS n'a pas écrite
  - à la fois parce qu'ils sont susceptibles de ne pas la connaître et parce que l'abstraction elle-même peut contenir des bogues/fonctionnalités qui ne sont pas présents dans le code RTOS de base.

# Comparaison entre FreeRTOS et CMSIS-RTOS

- On croit souvent à tort qu'il existe un RTOS appelé CMSIS-RTOS.
- CMSIS-RTOS n'est en fait qu'une définition de l'API.
- Son implémentation est en grande partie une couche de colle au RTOS sous-jacent, mais lorsqu'il existe des différences fonctionnelles entre les deux, un code de colle sera présent pour faire correspondre les fonctionnalités.

# Comparaison entre FreeRTOS et CMSIS-RTOS

- ARM a développé CMSIS-RTOS avec le même objectif que lors du développement de CMSIS : ajouter une couche d'abstraction cohérente qui réduit le verrouillage des fournisseurs.
  - Le CMSIS original était destiné à réduire le verrouillage des fournisseurs de silicium en fournissant des méthodes uniformes pour que les intergiciels accèdent aux fonctionnalités communes du Cortex-M. Il a atteint cet objectif.
  - Il a atteint cet objectif - il n'existe que quelques variantes de ports FreeRTOS pour les milliers de MCU à base de Cortex-M qu'il prend en charge.
  - De même, ARM tente maintenant de réduire le verrouillage des fournisseurs de RTOS en rendant le RTOS lui-même plus facile à changer - en fournissant une API cohérente (CMSIS-RTOS) qui n'est pas liée à un fournisseur.

# Comparaison entre FreeRTOS et CMSIS-RTOS

- Voici un aperçu rapide (plus de détails sont inclus dans la section Renvoi aux fonctions CMSIS-RTOS et FreeRTOS) :
  - **Tâches:** Il s'agit de la fonctionnalité permettant de créer et de supprimer des tâches avec des piles allouées de manière statique et dynamique.
  - **Sémaphores/mutex:** Les sémaphores binaires et de comptage ainsi que les mutex sont présents dans CMSIS-RTOS.
  - **Files d'attente:** Les API de files d'attente sont très similaires entre l'API native de FreeRTOS et l'API de CMSIS-RTOS.
  - **Minuteries logicielles:** Les API de temporisation logicielle sont très similaires entre l'API native de FreeRTOS et l'API de CMSIS-RTOS.
  - **Groupes d'événements:** Ils sont utilisés pour synchroniser plusieurs tâches.
  - **Contrôle du noyau/planificateur**: Les deux API ont la capacité de démarrer/arrêter des tâches et de surveiller le système.

# Points à prendre en compte lors de la migration

- Il y a quelques différences notables entre la programmation avec l'API CMSIS-RTOS et celle avec l'API FreeRTOS.
- Les fonctions de création de tâches de CMSIS-RTOS prennent la taille de la pile en octets, contrairement aux mots de FreeRTOS.
- Ainsi, appeler xTaskCreate sous FreeRTOS avec une taille de pile de 128 mots équivaut à appeler CMSIS-RTOS osThreadNew avec un argument de 512 octets.

# Référencement croisé des fonctions CMSIS-RTOS et FreeRTOS

## Fonctions de délai

| CMSIS-RTOS | FreeRTOS | Notes |
|---|---|---|
| osDelay | vTaskDelay | osDelay est exprimé en ms ou en ticks, selon la documentation et les commentaires auxquels vous vous fiez. Veillez à vérifier votre implémentation CMSIS-RTOS de osDelay() si une fréquence Systick autre que 1 kHz est utilisée! |
| osDelayUntil | vTaskDelayUntil, xTaskGetTickCount | |

# Référencement croisé des fonctions CMSIS-RTOS et FreeRTOS

| CMSIS-RTOS | FreeRTOS | Notes |
|---|---|---|
| osThreadEnumerate | vTaskSuspendAll, uxTaskGetNumberOfTasks, uxTaskGetSystemState, xTaskResumeAll | Cette opération suspend le système et remplit un tableau de gestionnaires de tâches. |
| osThreadNew | xTaskCreateStatic, xTaskCreate | |
| osThreadResume | vTaskResume | |
| osThreadSetPriority | vTaskPrioritySet | |
| osThreadSuspend | vTaskSuspend | |
| osThreadTerminate | vTaskDelete | Si Heap1 est utilisé, cette fonction renvoie osError. |
| osThreadYield | taskYIELD | |

# Référencement croisé des fonctions CMSIS-RTOS et FreeRTOS

## EventFlags

| CMSIS-RTOS | FreeRTOS | Notes |
|---|---|---|
| oseventFlagsClear | xEventGroupsClearBits, xEventGroupGetBitsFromISR | |
| osEventFlagsDelete | vEventGroupDelete | |
| osEventFlagsGet | xEventGroupGetBits, xEventGroupGetBitsFromISR | |
| osEventFlagsNew | xEventGroupCreateStatic, xEventGroupCreate | |
| osEventFlagsSet | xEventGroupSetBits, xEventGroupSetBitsFromISR | |
| osEventFlagsWait | xEventGroupWaitBits | |

# Référencement croisé des fonctions CMSIS-RTOS et FreeRTOS

| CMSIS-RTOS | FreeRTOS | Notes |
|---|---|---|
| osTimerDelete | xTimerDelete | Si Heap1 est utilisé, cette fonction renvoie osError. Elle libère également le TimerCallback_t* utilisé par le timer à supprimer. |
| osTimerGetName | pcTimerGetName | |
| osTimerIsRunning | xTimerIsTimerActive | |
| osTimerNew | xTimerCreateStatic, xTimerCreate | Allocation automatique pour TimerCallback_t. |
| osTimerStart | xTimerChangePeriod | |
| osTimerStop | xTimerStop | |

# Référencement croisé des fonctions CMSIS-RTOS et FreeRTOS

- L'interface de système d'exploitation portable (POSIX) a été développée pour fournir une interface unifiée pour interagir avec les systèmes d'exploitation, rendant le code plus portable entre les systèmes.

- Au moment de la rédaction de ce document, FreeRTOS dispose d'une implémentation beta pour un sous-ensemble de l'API POSIX.

# Référencement croisé des fonctions CMSIS-RTOS et FreeRTOS

- En règle générale, les fonctions de threading, de files d'attente, de mutex, de sémaphores, de temporisation, de sommeil et certaines fonctions d'horloge sont mises en œuvre par le port.

- Cet ensemble de fonctionnalités couvre parfois suffisamment de cas d'utilisation dans le monde réel pour permettre le portage d'applications qui ont été écrites pour être conformes à POSIX sur un MCU supportant FreeRTOS.

- Gardez à l'esprit que FreeRTOS ne fournit pas de système de fichiers par lui-même sans logiciel intermédiaire supplémentaire, donc toute application nécessitant un accès au système de fichiers aura besoin de composants supplémentaires avant d'être fonctionnelle.