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

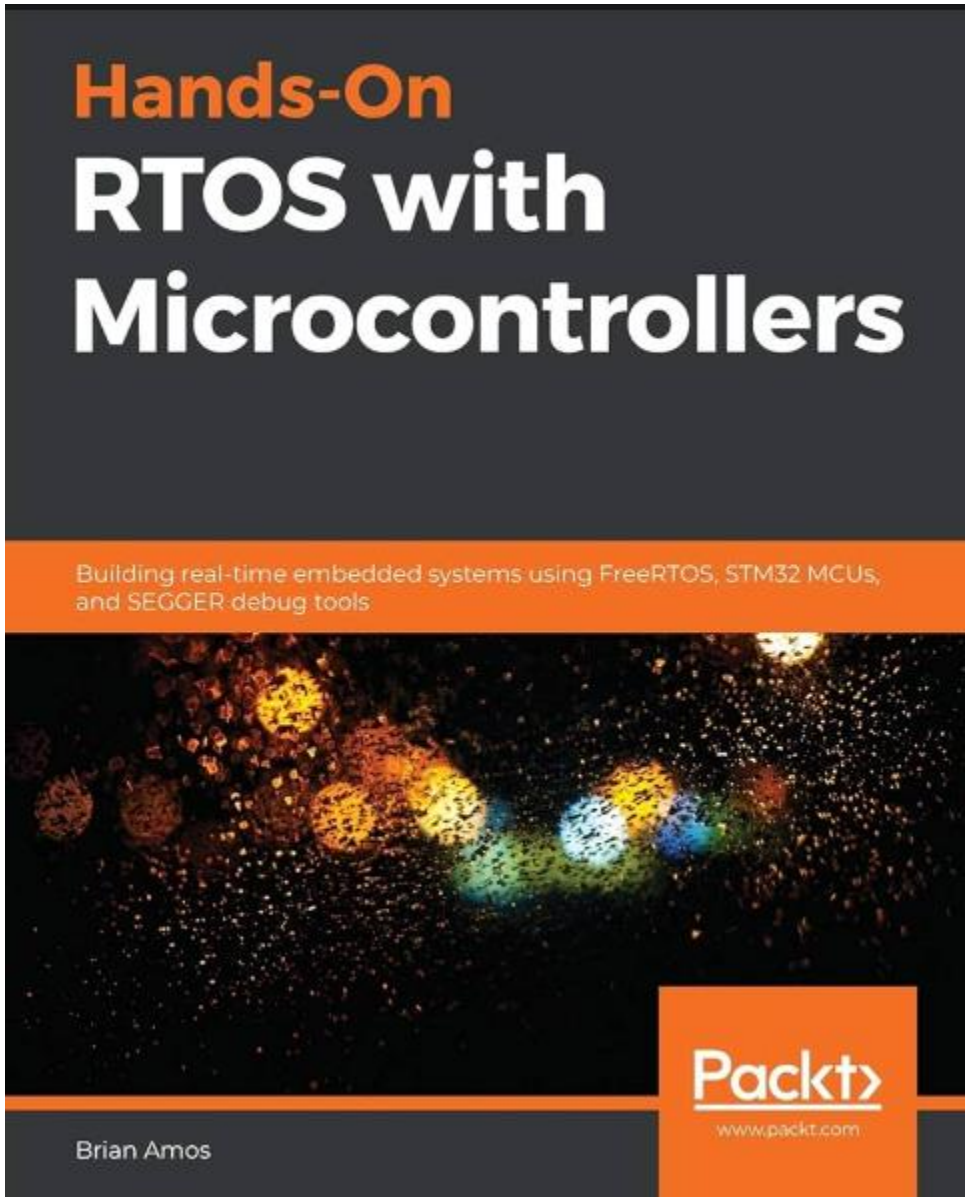
School of Electrical Engineering
and Computer Science

CEG4566/CSI4541/SEG4545
Conception de systèmes informatiques en temps réel

Winter 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 1:

Présentation des systèmes en temps réel

Plan

- Qu’est-ce que le "temps réel" ?
- Définition du système d'exploitation en temps réel (real-time operating system, RTOS)
- Décider quand utiliser un RTOS

Qu’est-ce que le temps réel ?

- Tout système qui a une réponse déterministe à un événement donné peut être considéré comme "temps réel."

- Si un système est considéré comme défaillant lorsqu’il ne répond pas à une exigence de synchronisation, il doit être en temps réel.

- La façon dont la défaillance est définie (et les conséquences d’un système défaillant) peut varier considérablement.

- Il est extrêmement important de se rendre compte que les exigences en temps réel peuvent varier considérablement, à la fois dans la vitesse de l’exigence de synchronisation et la gravité des conséquences si les délais en temps réel requis ne sont pas respectés.

Les plages d'exigences de synchronisation

- Pour illustrer la gamme des exigences de synchronisation qui peuvent être rencontrées, considérons quelques systèmes différents qui acquièrent des lectures de convertisseurs analogiques-numériques (*analog-to-digital converters, ADCs*).

- Le premier système que nous examinerons est un système de contrôle qui est mis en place pour contrôler la température d'un fer à souder (comme on le voit dans le diagramme suivant).

- Les parties du système qui nous intéressent sont le microcontrôleur (*microcontroller unit, MCU*), l'ADC, le capteur et le réchauffeur.

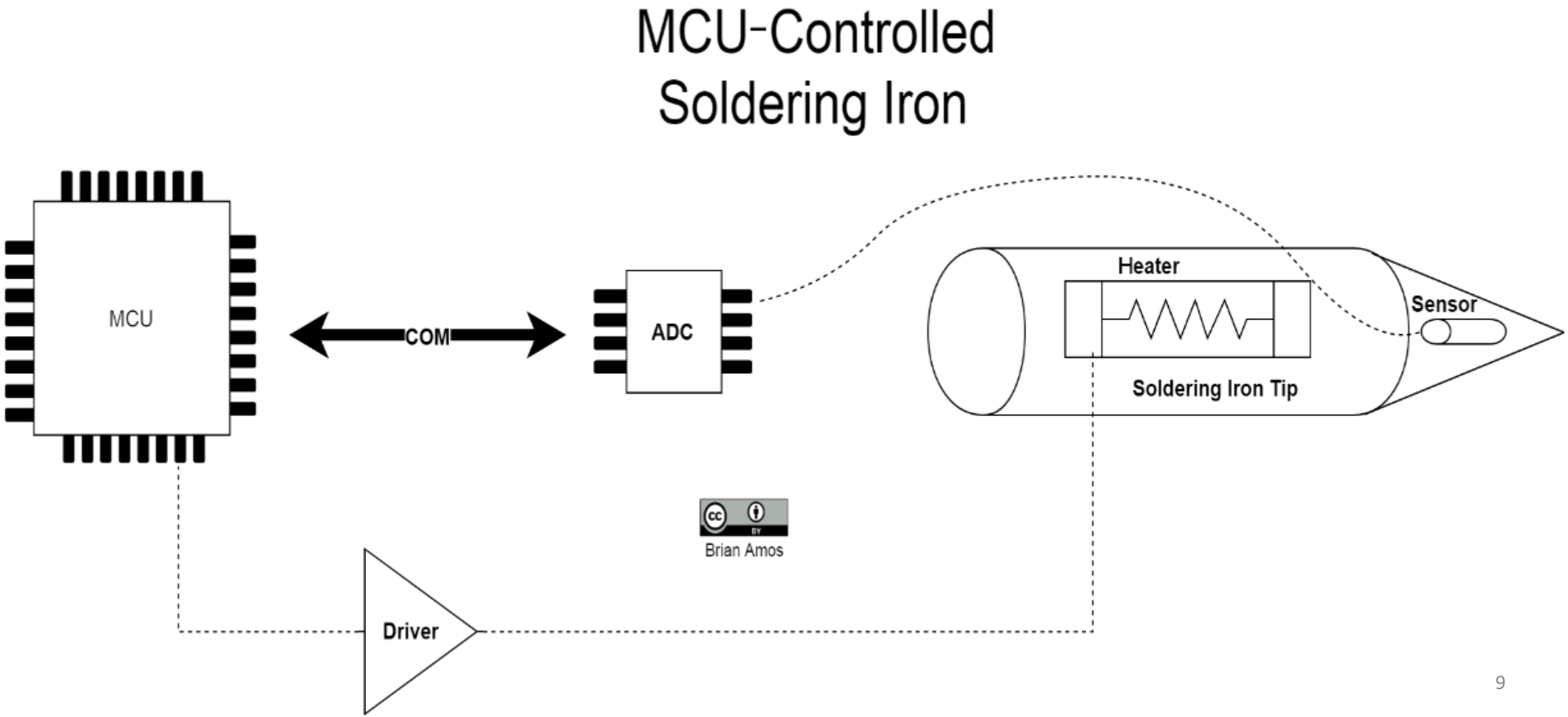
Le MCU est responsable de ce qui suit

- Prise de mesures à partir d'un capteur de température via l'ADC
- Exécution d'un algorithme de contrôle en boucle fermée (pour maintenir une température constante à la pointe du fer à souder)
- Réglage de la sortie de l'élément chauffant selon les besoins.

Le MCU de commande est responsable des tâches suivantes

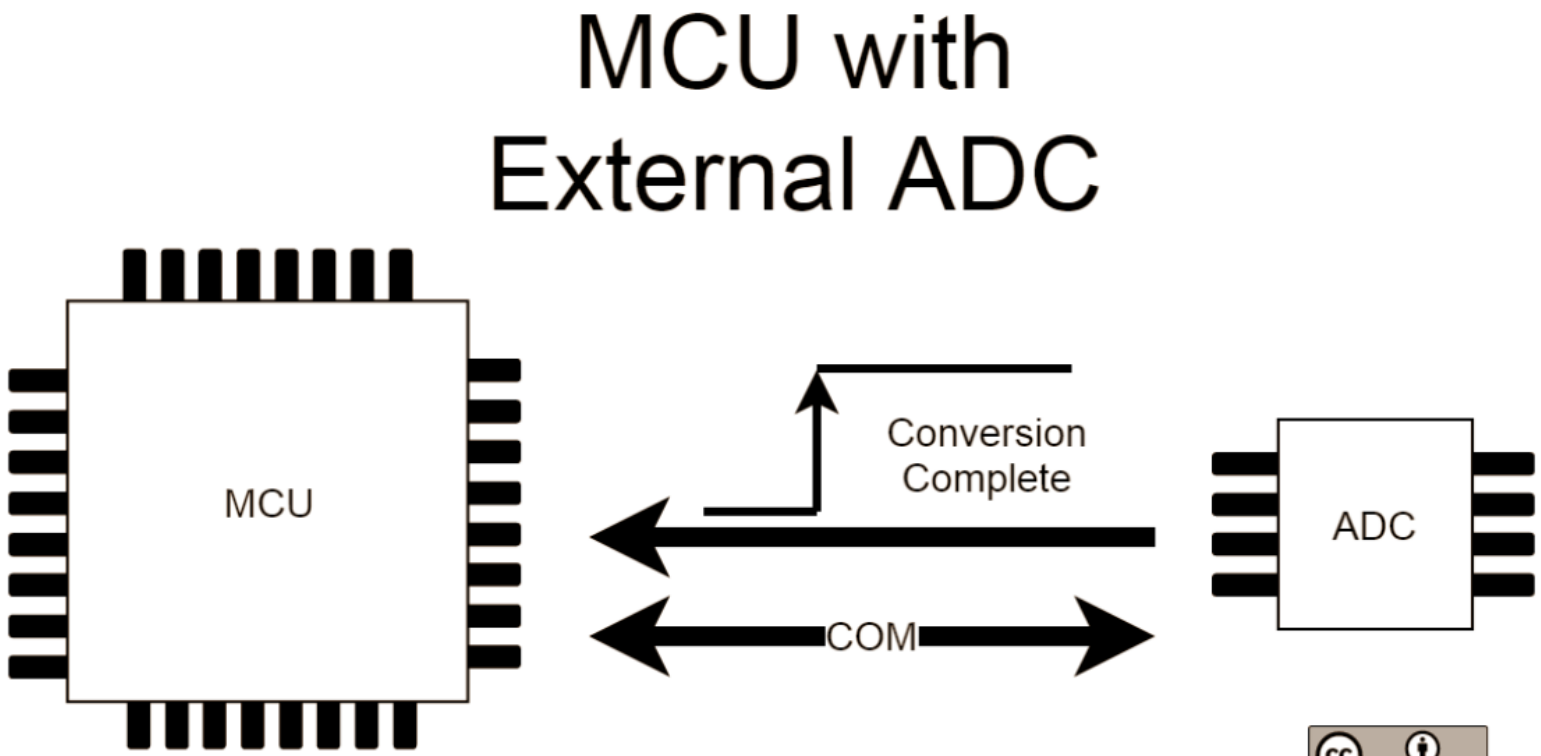
- Prise de mesures à partir d'un capteur de température via l’ADC.
- Exécution d'un algorithme de contrôle en boucle fermée (pour maintenir une température constante au niveau de la panne du fer à souder).
- Ajustement de la sortie de l'élément chauffant en fonction des besoins.

Ceux-ci peuvent être vus dans le diagramme suivant



Fer à souder commandé par MCU

- Comme la température de la pointe ne change pas très rapidement, le MCU peut n'avoir besoin d'acquérir que 50 échantillons ADC par seconde (50 Hz).
- L'algorithme de contrôle chargé d'ajuster le chauffage (pour maintenir une température constante) fonctionne à un rythme encore plus lent, 5 Hz :



Brian Amos

MCU lisant l’ADC

- L'ADC affirme une ligne matérielle, signalant qu'une conversion a été effectuée et qu'elle est prête pour que le MCU transfère la lecture dans sa mémoire interne.
- Le MCU qui lit le CAN a jusqu'à 20 ms pour transférer les données du CAN à la mémoire interne avant qu'une nouvelle lecture ne soit effectuée (comme le montre le diagramme suivant).
- Le MCU doit également exécuter l'algorithme de contrôle pour calculer les valeurs actualisées de la sortie du chauffage à 5 Hz (200 ms).
- Ces deux cas (bien qu'ils ne soient pas particulièrement rapides) sont des exemples d'exigences en temps réel :

Brian Amos

Les moyens de garantir un comportement en temps réel

- L'une des façons les plus simples de s'assurer qu'un système fait ce qu'il est censé faire est de s'assurer qu'il est aussi simple que possible tout en répondant aux exigences.

- Cela signifie résister à l'envie de trop compliquer une tâche simple.

- Si un grille-pain est censé griller une tranche de pain, il n'est pas nécessaire de l'équiper d'un écran pour qu'il vous indique également la météo; il suffit qu'il allume un élément chauffant pour le temps nécessaire.

- Cette tâche simple a été accomplie pendant des années sans avoir besoin de code ou de dispositif programmable.

Types de systèmes en temps réel

- Il existe de nombreuses façons d'obtenir un comportement en temps réel.
- La section suivante traite des différents types de systèmes en temps réel que vous pouvez rencontrer.
  - Matériel
  - Micrologiciel de type "bare-metal"
  - Micrologiciel basé sur un RTOS
  - Logiciel basé sur un RTOS
- Notez également qu'il est possible d'avoir des combinaisons des systèmes suivants fonctionnant ensemble en tant que sous-systèmes.
- Ces différents sous-systèmes peuvent être présents au niveau d'un produit, d'une carte ou même d'une puce.

Matériel

- Les systèmes en temps réel basés sur le matériel peuvent couvrir tous les domaines, des filtres analogiques aux codecs vidéo complexes, en passant par le contrôle en boucle fermée et les machines à états simples.

- Lorsqu'ils sont mis en œuvre dans un souci d'économie d'énergie, les composants intégrés spécifiques à une application (*application-specific integrated component, ASIC*) peuvent être conçus pour consommer moins d'énergie qu'une solution basée sur un MCU.

- En général, le matériel a l'avantage d'effectuer des opérations en parallèle et instantanément (il s'agit bien sûr d'une simplification excessive), contrairement à un MCU à cœur unique, qui ne donne que l'illusion d'un traitement parallèle.

Les inconvénients du matériel en temps réel

- Le manque de flexibilité des dispositifs non programmables.
- L'expertise requise est généralement moins disponible que celle des développeurs de logiciels/micrologiciels.
- Le coût des dispositifs programmables complets (par exemple, les grands FPGA).
- Le coût élevé du développement d'un ASIC personnalisé.

Micrologiciel de type "bare-metal"

- Les microprogrammes "bare-metal" sont considérés (pour nos besoins) comme des microprogrammes qui ne sont pas construits à partir d'un noyau/ordonnanceur préexistant d'un type ou d'un autre.

- Certains ingénieurs vont plus loin en affirmant qu'un véritable microprogramme "bare-metal" ne peut pas utiliser de bibliothèques préexistantes (telles que les bibliothèques d'abstraction matérielle fournies par les vendeurs), ce qui n'est pas le cas d'un microprogramme "bare-metal".

- Une implémentation "bare-metal" présente l'avantage que le code de l'utilisateur a le contrôle total de tous les aspects du matériel.

- Le seul moyen d'interrompre l'exécution du code de la boucle principale est de déclencher une interruption.

- Dans ce cas, le seul moyen pour qu'un autre élément prenne le contrôle du processeur est que l'ISR existant se termine ou qu'une autre interruption de priorité supérieure se déclenche.

Micrologiciel basé sur un RTOS

- Les microprogrammes qui exécutent un noyau d'ordonnancement sur un MCU sont des microprogrammes basés sur un RTOS.
- L'introduction de l'ordonnanceur et de certains primitifs RTOS permet aux tâches de fonctionner avec l'illusion qu'elles ont le processeur pour elles-mêmes.
- L'utilisation d'un RTOS permet au système de rester réactif aux événements les plus importants tout en exécutant d'autres tâches complexes en arrière-plan.
- L'exécution de toutes ces tâches présente quelques inconvénients.
- Des interdépendances peuvent apparaître entre les tâches qui partagent des données.
- si elles ne sont pas gérées correctement, ces dépendances peuvent entraîner le blocage inattendu d'une tâche.

Logiciels basés sur un RTOS (1/3)

- Les logiciels fonctionnant sur un système d'exploitation complet contenant une unité de gestion de la mémoire (memory management unit, MMU)) et une unité centrale de traitement (central processing unit, CPU)) sont considérés comme des logiciels basés sur un RTOS.

- Les applications mises en œuvre selon cette approche peuvent être très complexes et nécessiter de nombreuses interactions différentes entre divers systèmes internes et externes.

- L'avantage de l'utilisation d'un système d'exploitation complet réside dans toutes les capacités qui l'accompagnent, tant matérielles que logicielles.

Logiciels basés sur un RTOS (2/3)

- Sur le plan matériel, il y a généralement plus de cœurs de processeurs disponibles fonctionnant à des fréquences d'horloge plus élevées.

- Il peut y avoir des gigaoctets de mémoire vive (RAM) et de mémoire persistante disponibles.

- L'ajout de matériel périphérique peut être aussi simple que l'ajout d'une carte (à condition qu'il y ait des pilotes préexistants).

Logiciels basés sur un RTOS (3/3)

- Du côté des logiciels, il existe une multitude de solutions *open source* et propriétaires pour les piles de réseaux, le développement d'interfaces utilisateur, la gestion des fichiers, etc.

- Sous toutes ces capacités et options, le noyau est toujours implémenté de manière à ce que les tâches critiques ne soient pas bloquées pendant une période de temps indéfinie, ce qui est possible avec un système d'exploitation traditionnel.

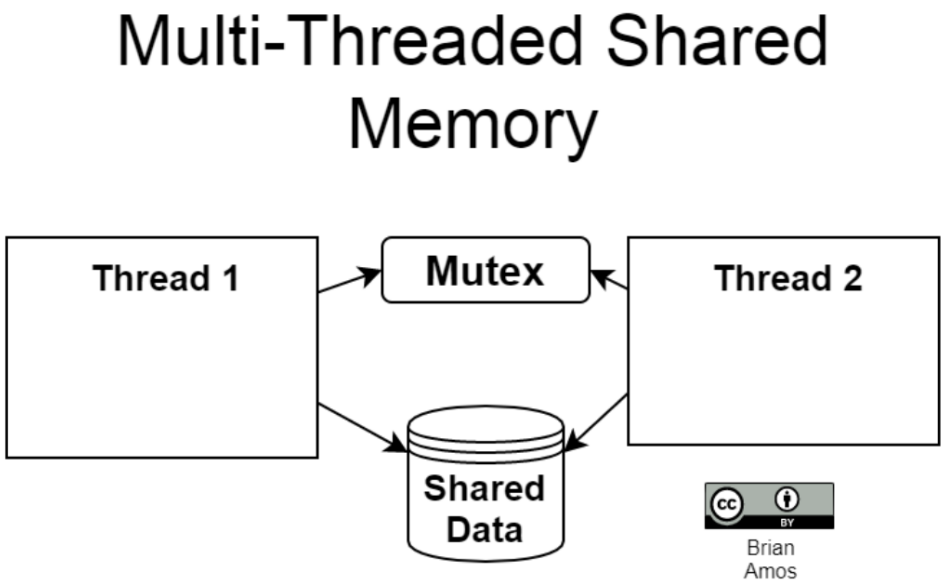
- De ce fait, il est toujours possible d'obtenir des performances déterministes, tout comme avec un microprogramme RTOS.

Définition du RTOS

- Les systèmes d'exploitation (tels que Windows, Linux et macOS) ont été créés pour fournir un environnement de programmation cohérent qui fait abstraction du matériel sous-jacent afin de faciliter l'écriture et la maintenance des programmes informatiques.

- Ils fournissent au programmeur d'applications de nombreuses primitives différentes (telles que les threads et les mutex) qui peuvent être utilisées pour créer un comportement plus complexe.

- Par exemple, il est possible de créer un programme multithread qui fournit un accès protégé à des données partagées :



Mémoire partagée multithread

- L'application précédente n'implémente pas les primitives de thread et de mutex, elle ne fait que les utiliser.
- Les implémentations réelles des threads et des mutex sont gérées par le système d'exploitation.
- Cela présente quelques avantages :
  - ✓ Le code de l'application est moins complexe.
  - ✓ Il est plus facile à comprendre
    - les mêmes primitives sont utilisées quel que soit le programmeur, ce qui facilite la compréhension d'un code créé par des personnes différentes.
  - ✓ La portabilité du matériel est meilleure
    - avec les précautions nécessaires, le code peut être exécuté sur n'importe quel matériel supporté par le système d'exploitation sans modification.

Système en temps réel dur (1/2)

- Un système en temps réel dur (critique) doit respecter son délai 100 % du temps.
- Si le système ne respecte pas un délai, on considère qu'il a échoué.
- Cela ne signifie pas nécessairement qu'une défaillance fera du tort à quelqu'un si elle se produit dans un système en temps réel dur
  - seulement que le système a échoué s'il ne respecte pas une seule échéance.

Système en temps réel dur (2/2)

- Les dispositifs médicaux, tels que les stimulateurs cardiaques et les systèmes de contrôle dont les paramètres sont contrôlés de manière extrêmement stricte, constituent des exemples de systèmes en temps réel difficiles à gérer.
- Dans le cas d'un stimulateur cardiaque, s'il ne parvient pas à administrer une impulsion électrique au bon moment, il risque de tuer le patient
  - c'est pourquoi les stimulateurs cardiaques sont définis comme des systèmes de sécurité critiques.
- À l'inverse, si un système de contrôle des mouvements sur une fraiseuse à commande numérique par ordinateur (*computer numerical control, CNC*)) ne réagit pas à temps à une commande, il peut plonger un outil dans la mauvaise partie de la pièce en cours d'usinage, la ruinant.
- Dans les cas que nous venons de citer, l'une des défaillances a entraîné la perte de vies humaines, tandis que l'autre a transformé du métal en ferraille
  - mais il s'agissait dans les deux cas d'échecs causés par un seul délai non respecté.

Systèmes en temps réel ferme

- Contrairement aux systèmes en temps réel dur, les systèmes en temps réel ferme doivent respecter leurs délais presque tout le temps.
- Si la vidéo et l'audio perdent momentanément leur synchronisation, cela ne sera probablement pas considéré comme une défaillance du système, mais perturbera probablement le consommateur de la vidéo.
- Dans la plupart des systèmes de contrôle (comme le fer à souder de l'exemple précédent), il est peu probable que quelques échantillons lus légèrement en dehors du temps spécifié détruisent le contrôle du système.
- Si un système de contrôle dispose d'un ADC qui prend automatiquement un nouvel échantillon, si le MCU ne lit pas le nouvel échantillon à temps, il sera écrasé par un nouveau.
- Cela peut se produire occasionnellement, mais si cela se produit trop souvent ou trop fréquemment, la stabilité de la température sera compromise.
- Dans un système particulièrement exigeant, il peut suffire de quelques échantillons manquants pour que l'ensemble du système de contrôle soit hors spécifications.

Systèmes en temps réel souple

- Les systèmes en temps réel non contraignants sont les plus souples en ce qui concerne la fréquence à laquelle le système doit respecter ses délais.
- Ces systèmes n'offrent souvent qu'une promesse de meilleur effort pour respecter les délais.
- Le régulateur de vitesse d'une voiture est un bon exemple de système en temps réel souple, car il n'y a pas de spécifications ou d'attentes strictes à son égard.
- Les conducteurs ne s'attendent généralement pas à ce que leur vitesse converge à +/- x mph/kph de la vitesse fixée.
- Ils s'attendent à ce que, dans des circonstances raisonnables, par exemple en l'absence de fortes pentes, le système de contrôle leur permette de se rapprocher de la vitesse souhaitée la plupart du temps.

La gamme des RTOS

- Les RTOS varient en fonction de leurs fonctionnalités, ainsi que de l'architecture et de la taille du processeur auquel ils sont le mieux adaptés.
- Du côté des plus petits, nous avons des RTOS 8-32 bits axés sur les MCU, tels que FreeRTOS, Keil RTX, Micrium µC, ThreadX, et bien d'autres.
- Cette catégorie de RTOS convient aux microcontrôleurs et fournit un noyau temps réel compact comme offre la plus basique.
- Lorsque vous passez des MCU aux processeurs d'application 32 et 64 bits, vous avez tendance à trouver des RTOS tels que Wind River VxWorks et Wind River Linux, Integrity OS de Green Hills, et même Linux avec les extensions de noyau PREEMPT_RT.
- Ces systèmes d'exploitation complets offrent une large sélection de logiciels, fournissant des solutions pour les exigences de planification en temps réel ainsi que pour les tâches informatiques générales.
- Même avec les systèmes d'exploitation que nous venons de citer, nous n'avons fait qu'effleurer la surface de ce qui est disponible. Il existe des solutions gratuites et payantes (certaines coûtant plus de 10 000 USD) à tous les niveaux de RTOS, grands et petits.

Le RTOS utilisé dans ce cours

- Avec toutes les options disponibles, vous vous demandez peut-être : pourquoi ce cours ne couvre-t-il qu'un seul RTOS sur un seul modèle de MCU ?
- Il y a plusieurs raisons à cela, la première étant que la plupart des concepts que nous allons couvrir sont applicables à presque tous les RTOS disponibles, de la même manière que les bonnes habitudes de codage transcendent le langage que vous êtes en train de coder.
- En nous concentrant sur une seule implémentation d'un RTOS avec un seul MCU, nous serons en mesure d'approfondir certains sujets, ce qui n'aurait pas été possible si toutes les alternatives avaient également été abordées.
- FreeRTOS est l'une des implémentations de RTOS les plus populaires pour les MCU et est très largement disponible.
- Il existe depuis plus de 15 ans et a été porté sur des dizaines de plateformes.
- Si vous avez déjà parlé à un ingénieur en systèmes embarqués de bas niveau qui est familier avec la programmation RTOS, il a certainement entendu parler de FreeRTOS et l'a probablement utilisé au moins une fois.
- En concentrant notre attention sur FreeRTOS, vous serez bien placé pour migrer rapidement votre connaissance de FreeRTOS vers d'autres matériels ou pour passer à un autre RTOS, si la situation l'exige.

Résumé

- Dans ce chapitre, nous avons abordé la manière d'identifier les exigences en matière de temps réel, ainsi que les différentes plates-formes disponibles pour la mise en œuvre de systèmes en temps réel.
- À ce stade, vous devriez être en mesure d'apprécier à la fois le large éventail de systèmes qui peuvent avoir des exigences en matière de temps réel, ainsi que la variété des moyens disponibles pour répondre à ces exigences en matière de temps réel.
- Dans le prochain chapitre, nous commencerons à approfondir les microprogrammes temps réel basés sur les MCU en examinant de plus près deux modèles de programmation différents
  - les super boucles et les tâches RTOS.