'Système embarqués et robotique'
Printemps 2024

Semaine 5

RTOS (Suite) et étude de cas de systèmes embarqués

AKEDDAR Hamza, KACK KACK Nathan, PETTINI Chiara

Contents

1 Architecture du ChibiOS, suite

1.1 Mécanismes du scheduler

Gérer à la main l'exécution de plusieurs fonctions concurrentes est possible, mais difficile. À la place, divers outils sont disponibles pour l'organisation des tâches du scheduler d'un RTOS. La suspension d'une tâche lui donne la liberté de restituer le contrôle au scheduler à un moment propice. À l'inverse, ce dernier peut ordonner l'arrêt d'une tâche pour en reprendre une autre, c'est la préemption.

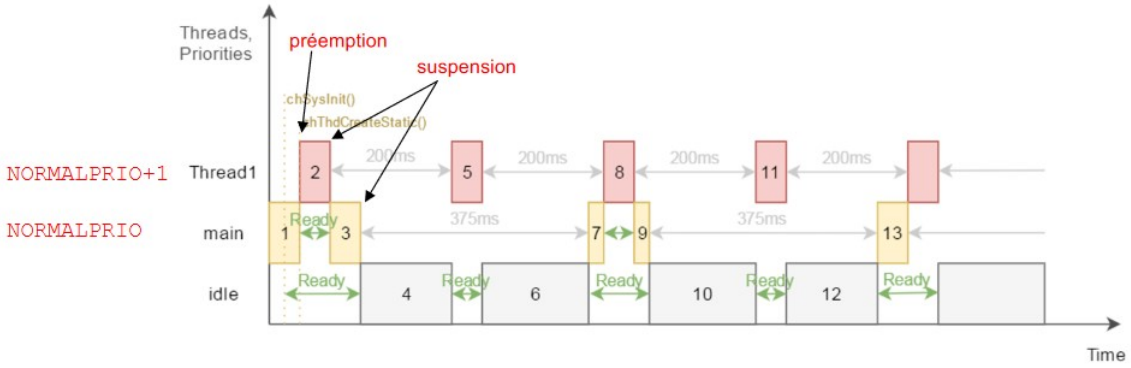

Figure 1: Illustration du scheduler (Cours 5 Slide 6)

Dans le cas où un certain thread a besoin d'accéder à une ressource critique en cours d'utilisation par un autre, nous pouvons utiliser les sémaphores ou les mutexes.

1.2 Sémaphores

Les sémaphores permettent d'assurer l'unité de l'accès à une ressource critique. Il en existe 2 types sur ChibiOS :

- Un sémaphore binaire est dans l'état "bloqué" tant qu'un Thread utilise une ressource. Lorsque cette ressource est libérée, le sémaphore passe en état "libre". Ce type de sémaphore permet de connaître le nombre de processus en attente.
- Un sémaphore avec compteur est utilisé lorsque de multiples ressources sont mises à disposition en parallèle. La valeur du compteur indique s'il y a des demandeurs en attentes, ou des ressources en surplus.

Le principal inconvénient des sémaphores est qu'ils ne permettent pas de connaître le propriétaire d'utilisation des ressources.

1.3 Mutexes

Les mutexes sont similaires aux sémaphores, mais permettent en plus de gérer l'héritage de priorité en intégrant la notion de prioritaire. Lorsque des tâches à haute priorité attendent une ressource, l'utilisation d'un mutex permet de faire hériter la haute priorité à une tâche qui utilise la ressource pour qu'elle se libère plus vite.

1.4 Mailboxes

Les mailboxes sont un moyen de communication entre threads. Chaque mailbox est associée à une file d'attente où les threads peuvent déposer et récupérer des messages. Lorsqu'un thread souhaite envoyer un message, il le place dans la mailbox correspondante qui fonctionne comme un buffer FIFO - First In First Out. Ce mécanisme permet le transfert asynchrone d'informations. Sans RTOS, pour communiquer des données entre routines, il faudrait utiliser des variables globales. Cette technique est largement déconseillée en raison de sa nature potentiellement dangereuse et prône aux bugs.

1.5 Condvar : Conditional variables / monitors

Les Condvar, ou variables conditionnelles, dans ChibiOS, sont des mécanismes additionnels pour gérer des conditions supplémentaires. Associées à un mutex, les condvar bloquent le thread jusqu'à ce qu' une condition spécifique ne soit satisfaite, libérant ainsi temporairement le mutex.

1.6 Events

Les Events offrent un moyen organisé de gérer différentes sources d'interruption. Ils peuvent répartir ses sources d'interruption parmi les tâches tout en contrôlant la façon d'attendre leur apparition et leur vérification.

1.7 Messages

Les messages, similaires au mailbox, sont une autre façon de gérer la communication entre threads. Il y a une hiérarchie entre les tâches: des clients et des serveurs. Les clients envoient un message et lancent une requête d'information. Les serveurs attendent qu'une demande arrive, la traitent et envoient la réponse.

2 Cycle de vie des sytèmes embarqués

Les systèmes embarqués comme les RTOS sont conçus afin d'optimiser l'utilisation de certaines ressources dans le but de répondre à des contraintes temporelles strictes. Avant d'incorporer un système embarqué dans un projet, il est essentiel de définir précisément les ressources nécessaires en fonction des exigences du projet. La première ressource cruciale pour n'importe quelle application est notre planète.

Chaque produit suit un cycle de vie généralement composé en 5 étapes bien distinctes: l'extraction des matériaux bruts, la production du produit, sa distribution, l'utilisation et enfin son recyclage.

Figure 2: Cycle de vie (presustainability.com)

L'empreinte d'un microcontrôleur peut être définie par différents indicateurs environnementaux tels que: le réchauffement climatique, la consommation d'eau, l'eutrophisation des milieux aquatiques et la formation d'Oxydants Photochimiques. Chaque étape du cycle de vie influence l'empreinte d'un système.

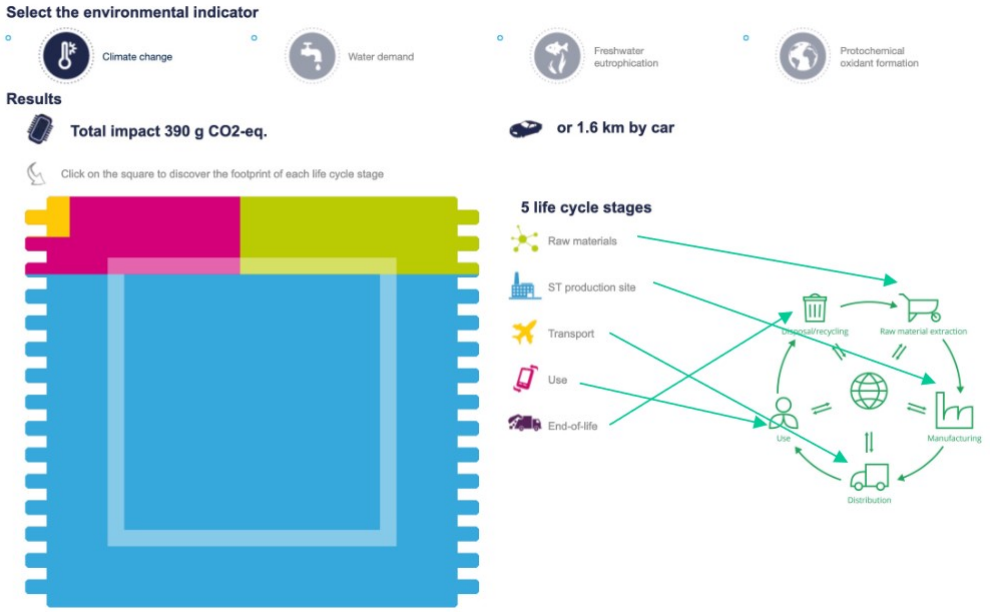

Figure 3: Empreinte carbone d'un microcontrôleur (Cours 5 Slide 30)

Vue que ces graphique montrent l'impact prioritaire de la fabrication, pour réduire l'impact environnemental, il est souvent plus judicieux de privilégier l'achat de biens d'occasion plutôt que de se concentrer uniquement sur des produits neufs classés A++. Cette approche permet de réduire la pression sur les ressources naturelles et de prolonger la durée de vie utile des produits.

En effet, chaque produit possède une proportion différente pour leur impact environnemental. Donc il est plus judicieux de se concentrer sur leurs étapes de vie ayant une influence plus importante.

3 Capteurs et contraintes

3.1 Calcul embarqué complexe

Nous prenons l'exemple d'une voiture haut de gamme, qui contient entre 100 et 150 processeurs effectuant de très différentes tâches ; contrôle moteur, gestion de frein, interface utilisateur... Il y aura donc une différence de niveaux de processeurs allant du microcontrôleur simple au processeur plus complexe que nous pouvons comparer, comme illustré dans la figure 5.

	i.MX31	dsPIC33
	Processeur	Microcontrôleur
Consommation	1W	300mW
MIPS	600	40
FPU	Oui	Non
Programming	Complex, OS	Simple
Packaging	BGA, 400 pins	TQFP, 100 pins
Prix	30$ + mémoire etc	5$ tout compris

Figure 4: Comparaison entre microcontrôleur et processeur (Cours 5 Slide 37)

En somme, il s'agit de bien choisir le bon processeur en fonction de l'application visée suivant ses caractéristiques.

3.2 Réseau de capteurs : Exemple : Station météo EPFL

En général, ils ont comme caractéristiques :

- Fréquence d'échantillonnage faible : < 1 Hz (par exemple chaque minute)
- Consommation très faible : 25 mW
- Alimentation solaire
- Transmission radio

Pour ces caractéristiques-ci, nous utilisons un processeur avec Cortex $M0$ qui consomme très faiblement dans ses différents modes (par exemple : 0.27 μA en Standby mode). Ces modes ne diffèrent que par leur spécificité de faible consommation ; avec ou sans Real Time Clock, toujours pour optimiser la consommation.

Afin de toujours moins consommer, il existe des méthodes mise en oeuvre pour entrer dans le Sleep mode, comme le WFI (Wait for interrupt) qui permet de rentrer immédiatement en sleep mode, ou le WFE (Wait for event) qui vérifie la valeur d'une registre avant de passer en sleep mode.

3.3 Monitoring accélération : Détection de chute pour appareils sensibles

En général, ces capteurs d'accélération 3 axes ont les caractéristiques suivantes :

- Fréquence d'échantillonnage basse : 1 kHz
- Acquisition A/D dans composant
- Consommation pas cruciale
- Embarqué dans un appareil

Nous prenons l'exemple de la chute d'un appareil de 56 cm, ce qui laisse 228 ms de possible activité afin de préserver les données avant l'impact avec le sol.

Le capteur d'accélération va opérer entre 500 et 4000 Hz en sortie et dispose de convertisseurs analogiques/numériques qui transmettent directement l'information au processeur.

3.4 Détection de provenance du son

Un capteur sonore possède des contraintes bien différentes que les autres. Une application de ce type de capteur est un robot qui se tourne dans la direction du son. L'e-puck2 possède 4 micros et permet de trianguler la direction du son en fonction du déphasage de celui-ci entre les microphones. Afin d'accomplir cette objectif il est nécessaire de répondre à des caractéristique précises:

- Fréquence d'échantillonnage : 44 kHz pour une bonne qualité audio
- Digitaliser le son sur le chip micro
- Décoder ce signal sur le microcontrolleur STM32F4

La reconstruction du signal prend 15% du CPU au total pour les 4 microphones, c'est une application qui demande beaucoup de capacité du microcontrôleur.

3.5 Vision pour détection automatique

Une autre ressource utilisable sont les caméras, celles-ci peuvent être utilisées dans divers applications comme dans la détection automatique pour les systèmes de surveillance. L'utilisation de ce type de ressource est caractérisée par:

- Flux de donnée très important de l'ordre du MHz
- Traitement de donnée conséquent
- Besoin considérable de mémoire RAM pour le stockage
- Vaste consommation selon les applications

L'acquisition totale d'une image est difficile voir impossible dans certains cas. Avec un simple calcul on peut voir qu'une image en couleur RGB (8,8,8 bits) de 640px x 480px prend 922 KBytes en comparaison l'e-puck2 ne possède que 192 KBytes de RAM.

La caméra de l'e-puck2 suit le schéma de la figure 5.

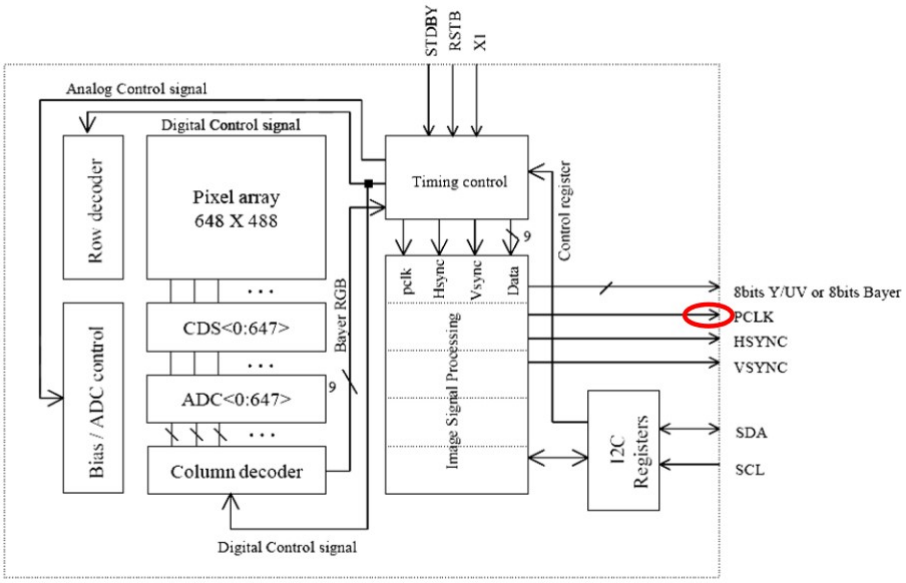

Figure 5: Schéma bloc de la caméra e-puck2

La pixel clock en sortie fournit l'image pixel par pixel, HSYNC permet de récolter ces pixels sur des lignes de dimension fixe et VSYNC permet de stacker ces lignes jusqu'à l'obtention de l'image entière. Le STM32 possède un digital camera interface (DCMI) qui est chargé de la réception et acquisition des images.