


Systèmes embarqués temps réel

Par: Youssef ROCHDI
youssefrochdi@yahoo.fr

Systèmes et réseaux temps réel
(youssefrochdi@yahoo.fr)

1



Plan du cours


- I. Introduction aux systèmes temps réel
- II. Traitement multitâches
- III. Gestion des tâches
- IV. Gestion des files
- V. Gestion des interruptions
- VI. Gestion des ressources
- VII. Gestion de la mémoire
- VIII. Résolution des problèmes

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

2



Activités pratiques


En parallèle avec le cours :

- I. Travaux dirigés et Travaux pratiques sur simulateur pour concrétiser les concepts fondamentaux.
- II. Travaux pratiques pour systèmes embarqués temps réel en utilisant les cartes:
 - USB PIC32 Starter kit II (www.Microchip.com)
 - Ou Carte Easy fusion de (www.Mikroe.com)
 - LCP17XX (www.NXP.com)
- III. Mini-projets/ exposés par groupe

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 3




Prérequis

- Etre familier avec la programmation modulaire en langage C.
- Avoir déjà suivi un cours d'info industrielle (microP, microC): de préférence microC de Microchip.
- Avoir des notions de base concernant les contraintes auxquelles sont soumis les systèmes embarqués.
- Etre capable de lire et comprendre des documents constructeurs (datasheets), manuel d'utilisateur (User manuel and Application Notes): en anglais.

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 4



Références

Le cours utilise l'executif temps réel freertos, lien web:
www.freertos.org (code source + documentation)

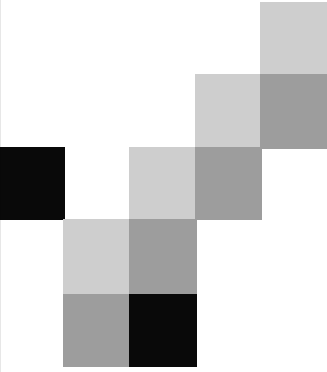
Bibliographie

- «Using the FreeRTOS Real Time Kernel - A Practical Guide»,
Richard Barry, ebook
www.freertos.org/Documentation/RTOS_book.html
- « C/OS, The Real-Time Kernel ». *Jean Labrosse, ebook,*
www.micrium.com
- « Real-Time Concepts for Embedded Systems ». *Qing Li with
Caroline Yao Published by CMP Books.*
- « Linux for Embedded and Real-time Applications ». *Doug Abbott,*
published by Newnes.
- « Systèmes d'exploitation ». *Andrew Tanenbaum. Edition
nouveaux horizons.*

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 5



Systèmes et réseaux temps réel

Chap I: Introduction

Systèmes et réseaux temps réel
(youssefrochdi@yahoo.fr)

6

1- Définition d'un STR (Real time System)

Un système temps réel (STR) est un système dont le fonctionnement est supposé correct (acceptable) ssi:

- Il possède un comportement entrées/sorties correct.
- En plus, les sorties produites doivent respecter certaines contraintes temporelles: échéances (DeadLines).

Il ne suffit pas que le système fournisse la bonne réponse (la bonne réaction), suite à des événements reçus, mais il faut que cette réponse soit fournie au "bon moment". Autrement le système peut devenir d'aucune utilité pour l'utilisateur.

2- Exemples de STR

Exemple 1 : Système automatique de péage sur autoroute par Tag RFID.

Si le système prend 2mn pour identifier un Tag (donc un véhicule), vérifier le solde, le système devient inutile et crée des embouteillages dans les stations de péage sur autoroutes.

Autant alors utiliser un système de péage manuel !!

On dit que le péage se fait " en temps réel " → sans arrêt de voiture

Moins de 5sec le système est fonctionnel mais la voiture doit réduire sa vitesse



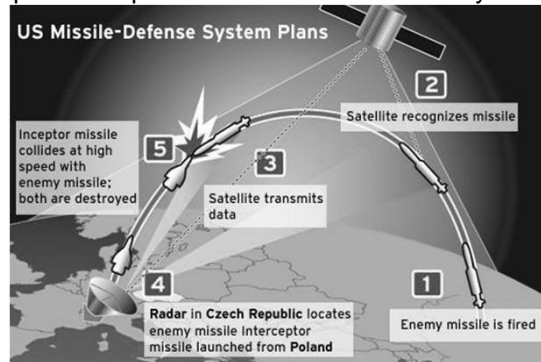
2- Exemples de STR

Exemple 2 : Système de défense anti-missiles.

Le radar permet de localiser le missile et de fournir sa position au système de contrôle qui calcule et ajuste l'angle de tir du missile intercepteur.

Si le bloc de contrôle prend 10sec pour calculer et ajuster l'angle de tir, le tir sera raté (missile ennemi est en mouvement à grande vitesse!).

Le système doit prédire la position future du missile → système prédictible



Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

9

2- Exemples de STR

Exemple 3 : Système informatisé de réapprovisionnement dans un supermarché.

Le système doit contrôler en permanence (en temps réel) les stocks et leur évolutions et doit émettre des commandes de réapprovisionnement aux fournisseurs pour maintenir les stocks entre de limites (Min, Max).

Si le système ne prend pas en considération les délais de livraisons, la dynamique d'évolution des stocks et les dates de péremption des produits, il y a un risque de rupture de stocks, ou de stocks remplis de produit périmés !

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

10

2- Exemples de STR

Exemple 4 : Ordinateur à bord d'une voiture.

Le système doit gérer plusieurs capteurs et contrôler plusieurs actionneurs au niveau du moteur et fournir les fonctionnalités d'aide à la conduite.

Entre autres, le sous-système de déclenchement des airbags est critique, car la sécurité des passagers en dépend.

Suite à un choc brutal, le système doit réagir dans les fractions de seconde qui suivent, indépendamment de la tâche qu'il effectuait au moment du choc.

Les tâches garantissant la sécurité doivent être prioritaires devant les autres tâches et le système doit garantir leur exécution aux bons moments.

Le système doit être préemptif (qu'on peut arrêter la tâche en cours pour aller exécuter une autre plus prioritaire). Il distingue les tâches critiques de celles non critiques.

3- Principales caractéristiques d'un STR

De ces exemples on peut faire les remarques suivantes:

- Tous ces systèmes doivent satisfaire des contraintes temporelles pour être utiles.
- Chacun de ces systèmes est amené à faire plusieurs tâches "au même temps". Certaines tâches sont plus prioritaires que d'autres.
- Un STR n'est pas forcément et toujours un système rapide: la gestion des accès concurrents aux ressources, la synchronisation sont aussi des aspects importants.
- Un STR ne doit pas répondre très tôt ou très tard il doit répondre juste à temps (JIT): Un système réactif idéal est Utopique! Il y a toujours des délais et de temps de calcul, tout dépendra de la tolérance des utilisateurs finaux.

3- Principales caractéristiques d'un STR

Ce qui est important à retenir:

- réagir en un temps adapté aux événements du monde réel,
- fonctionnement en continu sans réduire le débit du lot d'informations ou de données traitées,
- temps de calculs connus et modélisables pour permettre l'analyse de la réactivité.
- Latence maîtrisée.

Mécanismes à utiliser :

- horloges matérielles, interruptions, etc.
- multi-tâches, événements, messages, ordonnancement
- langages évolués voire spécifiques (langages synchrones, etc.)
- outils de modélisation: logique temporelle, réseaux de Petri, etc.

3- Types de STR

Un STR peut être de deux types:

- mou ou tolérant ou souple (Soft): le dépassement des échéances des actions peut être accepté occasionnellement et cause juste une dégradation des performances globales du système:

Système multimédia, téléphonie numérique

- Dur ou non tolérant ou strict (hard): le dépassement d'une échéance est catégoriquement non accepté car les conséquences peuvent être catastrophique et le système est, au meilleur des cas, non utile:

Système des Airbags, pancréas artificiels, etc.

Les échéances pour un Hard-RTS sont déterministes. Par contre pour un Soft-RTS elles peuvent ne pas l'être (mais dans une certaine plage de tolérance).

4- Domaines d'existence des STR

Les STR existent en différents domaines:

- ☐ Systèmes d'exploitation informatiques TR
- ☐ Systèmes de gestion de base de données TR.
- ☐ Systèmes audio/vidéo TR.
- ☐ Systèmes de télécoms TR.
- ☐ Systèmes de traduction/cryptage TR.
- ☐ Systèmes de télégestion TR.
- ☐ Systèmes de contrôle TR.

Ce cours se focalise, d'un point de vue technologique, sur les systèmes embarqués TR (SE/TR) et systèmes de contrôle/ supervision TR, distribués ou centralisés.

Les concepts de base restent valables quelque soit le domaine d'utilisation.

5- Conception d'un SETR

Système embarqué (embedded system) :

est un système offrant des services et comportant un calculateur (processeur) (MicroP, microC, DSP ϕ) en arrière plan, permettant d'exécuter des programmes spécifiques relatifs à ces services et installés par le concepteur lors de la fabrication;

à l'inverse des PC qui sont des calculateurs conçus pour exécuter différents programmes installés par l'utilisateur final.



5- Conception d'un SETR

Un système embarqué (SE) est souvent soumis à des contraintes sévères en terme de limitations au niveau de:

- Performances du calculateur: fréq processeur/ effet thermique.
- Autonomie en énergie: taille/poids batteries.
- Espace mémoire disponible: taille bus d'adresses/coût/espace

De plus un SE est basé sur un matériel (hardware) spécifique à sa finalité et une application (software) permettant son exploitation.

Ceci aura comme conséquences:

- le développement d'application temps réel est encore plus complexe pour le SE.
- L'application est, en général, non facilement portable sur un autre matériel.
- Il n'existe pas de méthode systématique pour le développement d'application temps réel.

5- Conception d'un SETR

Pour concevoir un SETR, deux approches possibles:

- 1ère approche:

Développement (de A à Z) d'application optimale, mais fortement dépendante du matériel.

- 2ème approche:

Développement (plus rapide) d'application basée sur un système d'exploitation ou noyau ou exécutif (Operating system, or Kernel, or executive) temps réel.

5- Conception d'un SETR

1^{ère} approche:

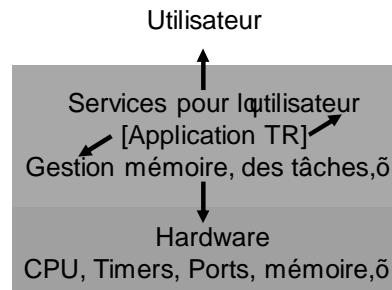
Inconvénients

- Le software est non portable.
- Grand effort de développement.
- Nécessite de bonnes connaissances du matériel et bcp de temps.

Avantages:

- Software optimisé pour le matériel: langage assembleur, économie de mémoire
- Totalement propriétaire, pas de frais de achat d'un middleware (software intermédiaire).

Valable pour les petits systèmes peu complexes, avec de faibles tailles de mémoire.



5- Conception d'un SETR

2^{ème} approche: basée sur Real time operating System (RTOS)

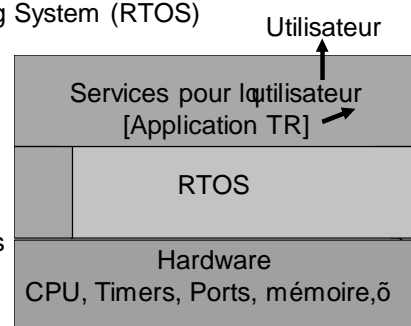
Avantages:

+ Le rtos, assure la gestion des ressources, l'interfaçage avec le hardware.

+ Le développement d'application TR est plus rapide et simple même pour des systèmes complexes (langage évolué comme C, Bibliothèques, mécanismes TR).

+ L'application est portable vers d'autres architectures matérielles supportées par le rtos choisi, maintenable et extensible.

+ La fiabilité de l'application est basée en grande partie sur celle du rtos.



5- Conception d'un SETR

2^{ème} approche: basée sur Realtime operating System (RTOS)

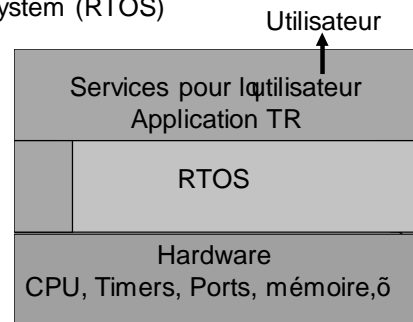
Inconvénients:

– coût supplémentaire du rtos (Licence, support technique).

– Le rtos consomme de la mémoire (footprint).

Cette approche est plus adaptée aux systèmes complexes avec moins de contraintes sur la mémoire,

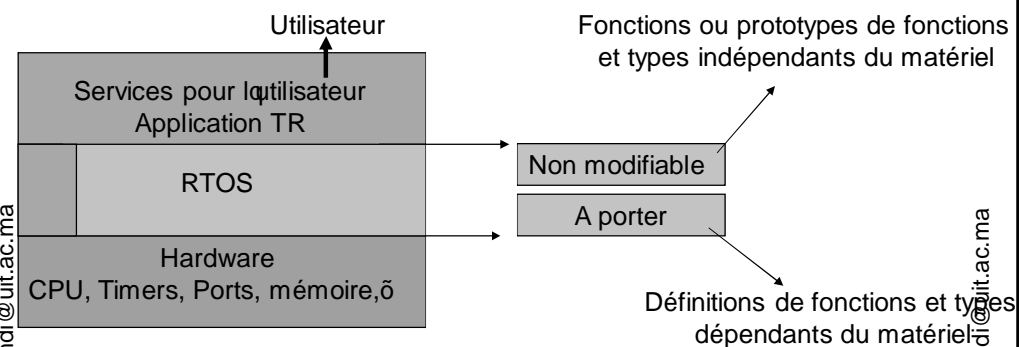
le coût du rtos peut être amorti par la rapidité et la facilité de développement, le respect des délais, la facilité de portabilité, la maintenance, et l'extension de l'application.



Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

21

5- Conception d'un SETR



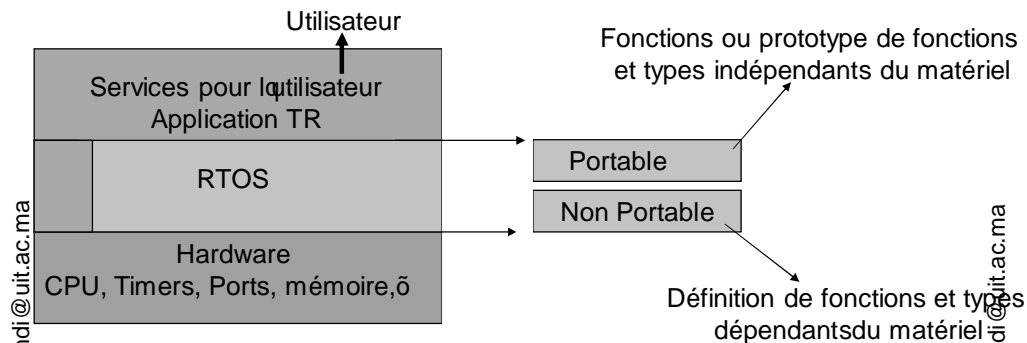
Conception du rtos en deux couches → implémentation sur diverses architectures matérielles.

La couche abstraite Non modifiable contient tous les mécanismes indépendants du matériel (ordonnancement, gestion de files, sémaphores...). La couche "A porter" ou nommée "portable" pour freertos, permet au Rtos de fonctionner sur une certaine architecture matérielle.

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

22

5- Conception d'un SETR



Plus intéressant, si un Rtos est déjà porté sur une architecture, le développeur se concentre sur les services que doit offrir son application.
Si le Rtos n'est pas déjà porté sur une architecture et le code du Rtos est disponible (opensource), le développeur est amené à porter le Rtos sur cette architecture avant de développer son application.

6- Quelques RTOS pour SETR

On donne ici quelques exemples de rtos qui ont été porté vers des architectures basées sur des microcontrôleurs PIC de Microchip.

Vendor/RTOS	Product	MPLAB® IDE Plug-in	Microchip 16/32 Ports	Modules/Support
AVIX	AVIX-RT 2.2.1	Yes	16/32	—
CMX Systems	CMX 5.30	Yes	8/16/32	TCP/IP
Express Logic	ThreadX G5.1.5.0	Yes	16/32	—
FreeRTOS	FreeRTOS v5.1.1	Yes	16/32	—
Micrium	μC/OS-II v2.84	Yes	8/16/32	TCP/IP
Pumpkin	Salvo 4 Pro and Salvo 4 LE	No	8/16/32	—
RoweBots	DSPNanoUnison	No	1632	DSP, TCP/IP, POSIX
Segger	embOS V3.52	Yes	16/32	GUI, TCP/IP

6- Quelques RTOS pour SETR

Ici un extrait de www.freertos.org, indiquant les architectures matérielles sur lesquelles freertos a été porté.

The 'Officially Supported' and 'Contributed' FreeRTOS Code page provides a detailed explanation of the differences between officially supported and contributed FreeRTOS ports. Officially supported FreeRTOS demos are provided that target microcontrollers from the following manufacturers:

- | | |
|-------------------|---------------------------------------|
| 1. Altera | 11. NEC |
| 2. Atmel | 12. Microsemi (formally Actel) |
| 3. Cortus | 13. NXP |
| 4. Cypress | 14. Renesas |
| 5. Energy Micro | 15. Silicon Labs |
| 6. Freescale | 16. ST Microelectronics |
| 7. Fujitsu | 17. Texas Instruments |
| 8. Infineon | 18. Xilinx |
| 9. Luminary Micro | 19. x86 (real mode) |
| 10. Microchip | 20. x86 / Windows Simulator |
| | 21. Unsupported and contributed ports |

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

6- Quelques RTOS pour SETR

Ici un extrait de www.micrium.com, indiquant les architectures matérielles sur lesquelles μ C/OS II et III ont été portés.

Download Center

Micrium's kernels have been ported to a wide variety of CPU architectures. **Browse by Semiconductor Vendor**

We provide not only the ports themselves, but also example projects for popular evaluation platforms. Each project is an ideal starting point for a new μ C/OS-based application.

Commercial Use: The example projects are intended for evaluation use only. Developers seeking to use Micrium products commercially must contact Micrium sales to obtain the proper license(s).

Ports and Drivers Not Listed: If you are interested in a port or driver for a hardware platform that is not listed here, you can [contact Micrium](#) to inquire about the possibility of commissioning a custom port.

- | | |
|---|--------------------------------------|
| • Altera | • MIPS Technologies |
| • Analog Devices | • NXP Semiconductors |
| • Atmel | • Renesas |
| • Cypress Semiconductor | • Samsung |
| • Freescale | • STMicroelectronics |
| • Fujitsu | • Tensilica |
| • Energy Micro | • Texas Instruments |
| • Infineon | • Toshiba |
| • Microchip | • Xilinx |
| • Microsemi | |


7- FREERTOS (1)

Le cours se basera sur freertos en utilisant
L'architecture matérielle 24 bits et 32 bits de Microchip:

1- PIC32 USB Starter kit II basé sur PIC32MX795F512L

TABLE 3: PIC32 USB, ETHERNET AND CAN – FEATURES

Device	Pins	Program Memory (KB)	Data Memory (KB)	USB, Ethernet and CAN									
				USB	Ethernet	CAN	Timers/Capture/Compare	DMA Channels (Programmable/Dedicated)	UART(2,3)	SPI(3)	I ² C™(3)	10-bit 1 Msps ADC (Channels)	Comparators
PIC32MX775F512L	100	512 + 12 ⁽¹⁾	64	1	1	2	5/5/5	8/8	6	4	5	16	2
PIC32MX795F512L	100	512 + 12 ⁽¹⁾	128	1	1	2	5/5/5	8/8	6	4	5	16	2



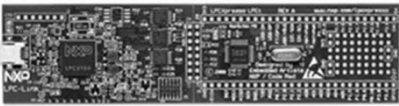
PIC32 USB Starter Kit II


Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 27

7- FREERTOS (2)


Une démo sur une architecture NXP
2-NXP LPCXpresso LPC1769

The LPC1769 is a Cortex-M3 microcontroller for embedded applications featuring a high level of integration and low power consumption at frequencies of 120 MHz. Features include 512 kB of flash memory, 64 kB of data memory, Ethernet MAC, USB Device/Host/OTG, 8-channel DMA controller, 4 UARTs, 2 CAN channels, 3 SSP/SPI, 3 I2C, I2S, 8-channel 12-bit ADC, 10-bit DAC, motor control PWM, Quadrature Encoder interface, 4 general purpose timers, 6-output general purpose PWM, ultra-low power Real-Time Clock with separate battery supply, and up to 70 general purpose I/O pins. The LPC1769 is pin-compatible to the 100-pin LPC2368 ARM7 MCU





LPC-Link




Target


Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 28

Youssef.rochdi@uit.ac.ma

7- FREERTOS (3)



→ ↻ 🏠
🔒 https://www.freertos.org 80% ... ☆
⬇ 📄



Quality RTOS & Embedded Software
About Contact Support FAQ Download

Quick Start Supported MCUs PDF Books Trace Tools Ecosystem Email List

Home
MIT License
FreeRTOS Books and Manuals
FreeRTOS
FreeRTOS Interactive

Quick Start Guide
Support Forum
Download Source

FreeRTOS+ Ecosystem
FreeRTOS+TCP: Thread safe TCP/IP stack
SafeRTOS: TUV certified RTOS
OpenRTOS: Commercial Licensed RTOS
Fail Safe File System: Ensures data integrity
FreeRTOS BSPs: 3rd party driver packages

The FreeRTOS™ Kernel

Market Leading, De-facto Standard and Cross Platform RTOS kernel.

Immediate Free Download and Use • Feature Rich • Tiny Footprint • Easy To Use Pre-configured Projects • Can Be Used in Commercial Applications • Massive User Community • Free Support • Optional Commercial Licensing/Support • Strict Coding Standard • Safety Critical Version Available • Tickless Mode for Low Power Applications

Developed in partnership with the world's leading chip companies over a 15 year period, the FreeRTOS kernel is a market leading real time operating system (or RTOS), and the de-facto standard solution for microcontrollers and small microprocessors.

With millions of deployments in all market sectors, blue chip companies trust FreeRTOS because it is professionally developed, strictly quality controlled, robust, supported, free to use in commercial products without a requirement to expose proprietary source code, and its IP is carefully managed.

The FreeRTOS value proposition . . .
Things you might not know about FreeRTOS . . .
Why choose FreeRTOS? . . .

Latest News

NXP tweet showing LPC5500 (ARMv8-M Cortex-M33) running FreeRTOS.

Meet Richard Barry and learn about running FreeRTOS on RISC-V at FOSDEM 2019



Version 10.1.1 of the FreeRTOS kernel is available for immediate download. MIT licensed.

View a recording of the "OTA Update Security and Reliability" webinar, presented by TI and AWS.

Careers

FreeRTOS and other embedded software careers at AWS.

FreeRTOS Partners


 

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

29

Youssef.rochdi@uit.ac.ma

7- FREERTOS (4)



- Code source de Freertos est organisé en deux parties:
 - Non modifiable (commune à toutes les architectures matérielles)

```


FreeRTOS
├── Source
│   ├── tasks.c           FreeRTOS source file - always required
│   ├── list.c            FreeRTOS source file - always required
│   ├── queue.c           FreeRTOS source file - nearly always required
│   ├── timers.c          FreeRTOS source file - optional
│   ├── event_groups.c    FreeRTOS source file - optional
│   └── croutine.c        FreeRTOS source file - optional

```

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

30

7- FREERTOS (5)



- Code source de Freertos est organisé en deux parties:
 - Adaptable au matériel (A porter))

```


FreeRTOS
├── Source
│   ├── portable Directory containing all port specific source files
│   ├── MemMang Directory containing the 5 alternative heap allocation source files
│   ├── [compiler 1] Directory containing port files specific to compiler 1
│   │   ├── [architecture 1] Contains files for the compiler 1 architecture 1 port
│   │   ├── [architecture 2] Contains files for the compiler 1 architecture 2 port
│   │   └── [architecture 3] Contains files for the compiler 1 architecture 3 port
│   ├── [compiler 2] Directory containing port files specific to compiler 2
│   │   ├── [architecture 1] Contains files for the compiler 2 architecture 1 port
│   │   ├── [architecture 2] Contains files for the compiler 2 architecture 2 port
│   │   └── [etc.]

```

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

31

7- FREERTOS (6)




- Fournit une solution pour de nombreuses architectures et outils de développement.
- Est connu pour être fiable. La confiance est assurée par les activités entreprises par le projet frère SafeRTOS.
- Est en cours de développement actif et continu.
- Nécessite une ROM, RAM minimale, et surcharge minimale de traitement. Typiquement, une image binaire du noyau RTOS sera dans la plage de 4K à 9K octets.
- Est très simple - le noyau du RTOS est contenu dans seulement 3 fichiers C. La majorité des nombreux fichiers inclus dans le téléchargement du fichier .zip ne concernent que les nombreuses applications de démonstration.
- Est vraiment gratuit pour une utilisation dans des applications commerciales (voir les conditions de licence pour plus de détails).

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

32

7- FREERTOS (7)

- Livré avec un portage, une plate-forme de développement ou un service de développement d'applications si nécessaire.
- Très utilisé avec une base d'utilisateurs importante et grandissante. Contient un exemple préconfiguré pour chaque port.
- Pas besoin de comprendre comment configurer un projet - il suffit de télécharger et de compiler!
- A un excellent forum de support gratuit, surveillé et actif.
- A l'assurance qu'un soutien commercial est disponible, le cas échéant.
- Fournit une documentation complète. Est très évolutif, simple et facile à utiliser.
- FreeRTOS offre une alternative de traitement en temps réel plus petite et plus facile pour les applications où eCOS, Linux embarqué (ou Real Time Linux) et même uCLinux ne conviennent pas, ne sont pas appropriés ou ne sont pas disponibles.



Youssef.rochdi@uit.ac.ma

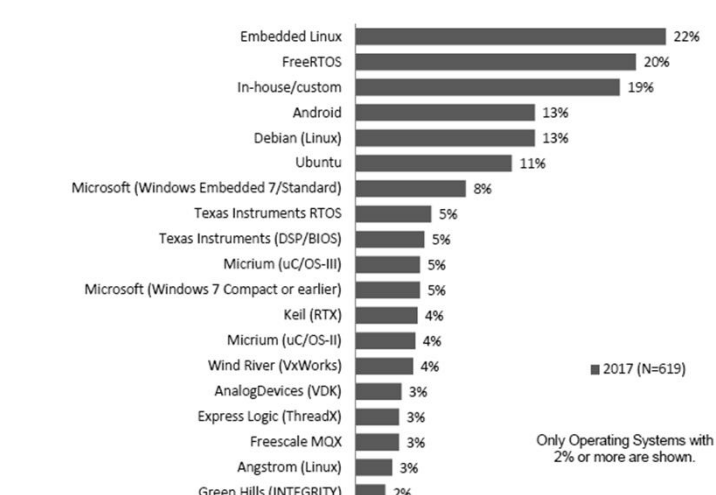
Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

Youssef.rochdi@uit.ac.ma

33

7- FREERTOS (8)


Please select ALL of the operating systems you are currently using.



Operating System	Percentage
Embedded Linux	22%
FreeRTOS	20%
In-house/custom	19%
Android	13%
Debian (Linux)	13%
Ubuntu	11%
Microsoft (Windows Embedded 7/Standard)	8%
Texas Instruments RTOS	5%
Texas Instruments (DSP/BIOS)	5%
Micrium (uC/OS-III)	5%
Microsoft (Windows 7 Compact or earlier)	5%
Keil (RTX)	4%
Micrium (uC/OS-II)	4%
Wind River (VxWorks)	4%
AnalogDevices (VDK)	3%
Express Logic (ThreadX)	3%
Freescale MQX	3%
Angstrom (Linux)	3%
Green Hills (INTEGRITY)	2%

■ 2017 (N=619)

Only Operating Systems with 2% or more are shown.




Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

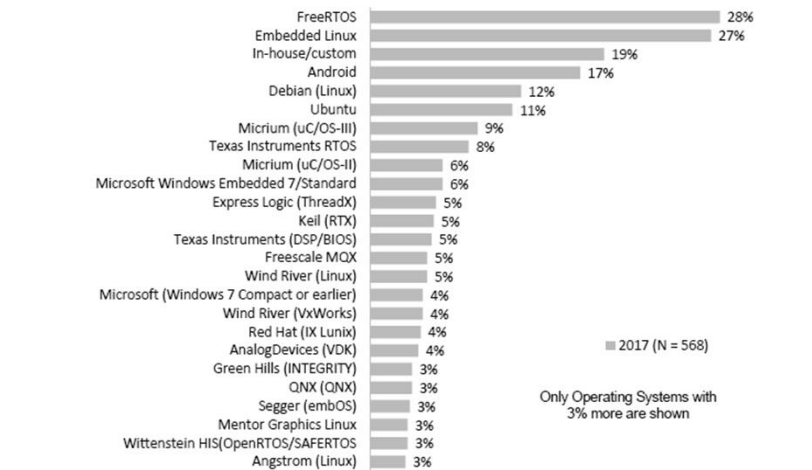
Youssef.rochdi@uit.ac.ma

34

7- FREERTOS (9)



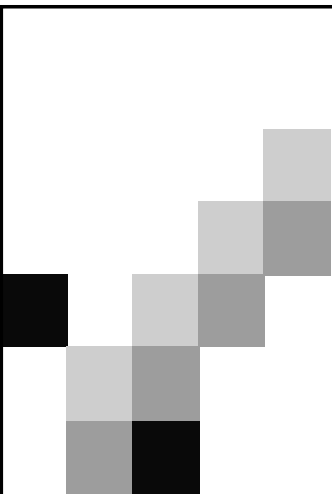
Please select ALL of the operating systems you are considering using in the next 12 months.



Operating System	Percentage
FreeRTOS	28%
Embedded Linux	27%
In-house/custom	19%
Android	17%
Debian (Linux)	12%
Ubuntu	11%
Micrium (uC/OS-III)	9%
Texas Instruments RTOS	8%
Micrium (uC/OS-II)	6%
Microsoft Windows Embedded 7/Standard	6%
Express Logic (ThreadX)	5%
Keil (RTX)	5%
Texas Instruments (DSP/BIOS)	5%
Freescale MQX	5%
Wind River (Linux)	5%
Microsoft (Windows 7 Compact or earlier)	4%
Wind River (VxWorks)	4%
Red Hat (IX Linux)	4%
AnalogDevices (VDK)	4%
Green Hills (INTEGRITY)	3%
QNX (QNX)	3%
Segger (embOS)	3%
Mentor Graphics Linux	3%
Wittenstein HIS/OpenRTOS/SAFERTOS	3%
Angstrom (Linux)	3%

■ 2017 (N = 568)

Only Operating Systems with 3% more are shown



Systèmes embarqués temps réel
Chap II: Traitement multitâches

Systèmes et réseaux temps réel
(youssefrochdi@yahoo.fr)

36

Objectifs

- Rappeler Ou expliquer le principe du traitement multitâches.
- Expliquer le rôle de l'ordonnanceur.
- Donner le Diagramme d'états des tâches.
- Citer les algorithmes d'ordonnement de freertos.
- Définir la tâche particulière Idle (état de "repos") et son intérêt.
- Conclure quand aux intérêts du traitement multitâches.

1- Définition

- Le traitement multi-tâches → Exécuter des programmes indépendants ou des parties de programmes relativement indépendantes par un seul processeur*, afin de donner l'illusion qu'ils s'exécutent en parallèle.

(ou un nombre limité de processeurs de type multi-cœurs)

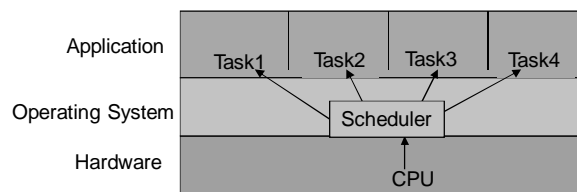
Technique qui date des premiers gros ordinateurs (supercalculateurs, chers, entourés de plusieurs terminaux → donc plusieurs utilisateurs et plusieurs programmes).

- Technique utilisée de nos jours dans la plus part des systèmes d'exploitation courants pour Systèmes informatiques, systèmes embarqués, systèmes industriels.



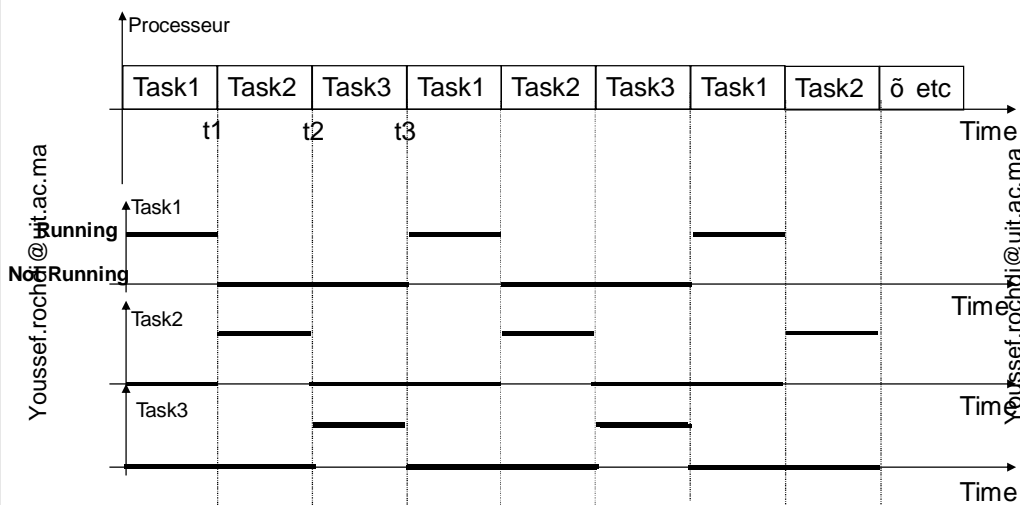
2- Ordonnanceur

- Une application est divisée en plusieurs tâches.
- Une tâche (Task) est une entité en exécution (Code et des données): c'est une occurrence de l'exécution d'un (ou d'une partie) programme.
- Le seul processeur disponible ne peut exécuter qu'une tâche à la fois. → Un composant du système d'exploitation, appelé «Ordonnanceur » (Scheduler) permet d'attribuer à l'aide d'un algorithme d'ordonnement le processeur aux tâches



3- Exemple simple d'ordonnement

- Exemple d'ordonnement (Chronogramme d'exécution des tâches)



4- Modèle simplifié des états d'une tâche

- A tout instant chaque tâche est dans l'un des deux états: en exécution (Running) et non exécutée (Not Running).

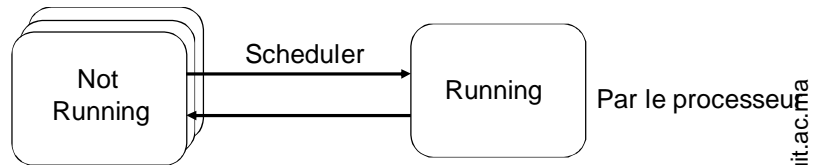
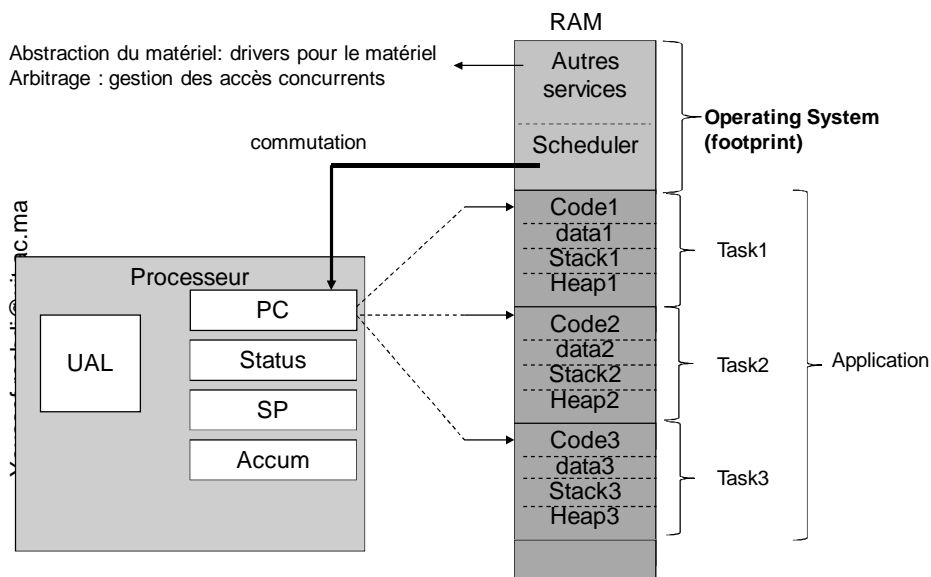


Diagramme simplifié des états d'une tâche

Une tâche peut être considérée comme indépendante, elle peut contenir des fonctions appelant d'autres fonctions; elle dispose alors de sa propre pile (stack) et d'un Heap (alloc-mém dynamiq).

- Une tâche est en état « Not Running », pour plusieurs raisons (voir plus loin).

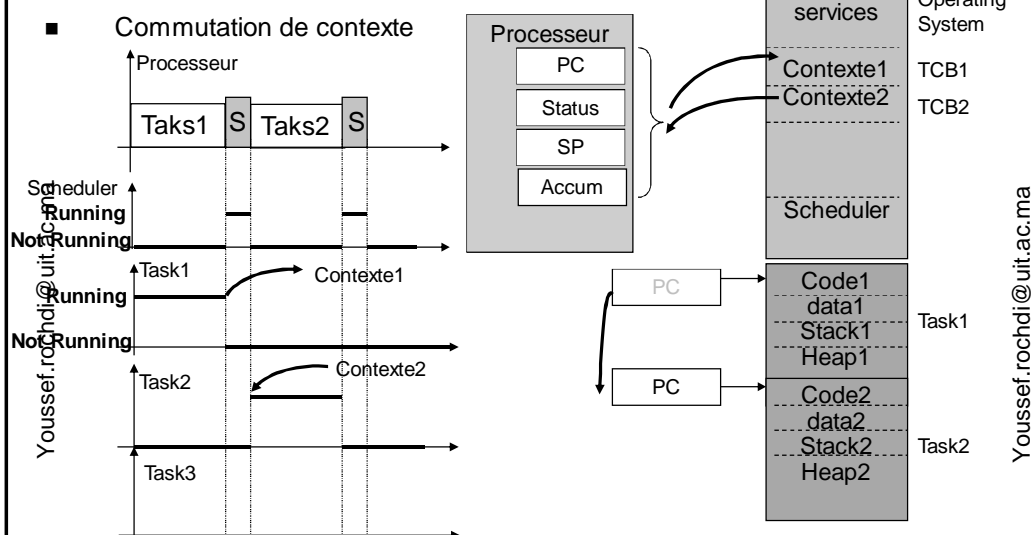
5- Organisation de la mémoire centrale




6- Commutation de contexte(1)

- Quand le scheduler décide d'abandonner par exemple tâche1 pour exécuter tâche2 il doit:
 - Mémoriser l'adresse de la prochaine instruction à exécuter dans tâche1 (contenu actuel du PC).
 - Mémoriser les contenus des registres du processeur (Stack Pointer, STATUS, Accumulators...).
 - Restaurer dans PC, l'adresse de la prochaine instruction à exécuter dans tâche 2, déjà mémorisée quand tâche 1 a été abandonnée.
 - Restaurer les contenus des registres du processeur, déjà mémorisés quand tâche 2 a été abandonnée.
- Cette opération s'appelle la commutation de contexte.
- Lors de la création d'une tâche le système d'exploitation lui réserve un bloc de mémoire appelé bloc de contrôle (TCB Task Control Block) pour sauvegarder son contexte et d'autres propriétés.

6- Commutation de contexte(2)





7- Algorithmes d'ordonnancement(1)


- freeRTOS utilise deux types d'algorithmes d'ordonnancement :
 - Préemptif:
 - Le scheduler s'exécute à chaque top (tick) d'horloge système et fait une commutation de contexte.
 - Le scheduler attribue donc une durée maximale d'exécution appelée quantum (tick) pour chaque tâche (horloge système).
 - Si passé ce quantum la tâche n'est pas encore terminée, elle est préemptée (interrompue) par le Scheduler qui choisit parmi les tâches prêtes en attente une autre tâche à poursuivre.
 - Si par contre la tâche s'est bloquée (entrée/sortie ou attente d'un message d'une autre tâche ou volontairement arrêtée) avant la fin du quantum, le scheduler prend le contrôle pour choisir une autre tâche prête à poursuivre.

NB: le choix de la tâche à exécuter se fait selon les priorités des tâches, à même priorité → FIFO (Round Robin).

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 45



7- Algorithmes d'ordonnancement(2)

- Non Préemptif ou collaboratif:

Un tâche s'exécute jusqu'à ce que:

 - elle cède d'elle-même le processeur par un appel d'une fonction explicite.
 - Elle rentre dans un état bloqué, par appel d'une fonction d'entrée/sortie ou en attente de synchronisation ou temporisation.
 - Elle est interrompue par une interruption.

Dans quel cas, le scheduler se manifeste pour choisir une autre tâche ou exécuter la routine d'interruption et faire une commutation de contexte.

Le choix entre les deux algorithmes se fait lors de la compilation et à l'aide du paramètre **configUSE_PREEMPTION** dans le fichier **freertosconfig.h**

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 46

8- Diagramme complet des états d'une tâche(1)

Quand une tâche n'est pas dans l'état Running, il est dans l'un des états suivants:

- Ready (Prête):

Prête à être exécuter, elle attend qu'elle soit choisie par le scheduler.

- Blocked (Bloquée):

lors qu'elle était en exécution, elle a fait appel à une fonction bloquante (entrée/sortie, temporisation, attente de synchronisation...). Une fois, la fonction bloquante terminée, la tâche revient systématiquement à l'état prête (ready).

- Suspended (Suspendue):

volontairement suspendue par appel à une fonction explicite, la tâche reste dans cet état jusqu'à ce qu'elle soit reprise par appel à une fonction explicite.

8- Diagramme complet des états d'une tâche(2)

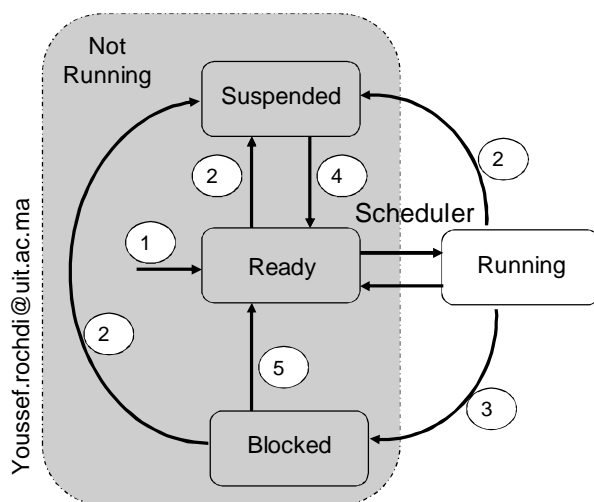


Diagramme Complet des états d'une tâche

9- Influence de l'algorithme d'ordonnement Etude de cas (1)

But:

- Etudier à travers un exemple l'influence du type d'ordonnement sur le comportement de l'application.
- Comprendre les transitions entre les états Ready, Running, Blocked

On a une application composée de 3 tâches Tsk1, Tsk2 et Tsk3, créées dans cet ordre. Chaque tâche effectue des calculs (C) et des opérations d'entrée/sortie (E/S) sur une même ressource d'E/S partagée, comme indiquée dans le tableau suivant:

Tâche	Comportement
Tsk1	(C) 30s → (E/S) 10s → (C) 25s
Tsk2	(C) 25s → (E/S) 20s → (C) 10s
Tsk3	(C) 10s → (E/S) 20s

On suppose que:

- la commutation de contexte se fait instantanément.

9- Etude de cas (2)

Algorithme préemptif: quantum(tick)=10s et même priorité

Running	1	2	3	1	2	1	2	i	1	1	2	1	i	i
Ready	2	3	1	2	1	2	v	v	v	2	1	v		
Blocked	v	v	v	3	3	v	1	1	2	2	v	v	v	

i: idle task, tâche particulière exécutée quand aucune autre tâche ne s'exécute.

V: file d'attente vide

Tâche	Comportement
Tsk1	(C) 30s → (E/S) 10s → (C) 25s
Tsk2	(C) 25s → (E/S) 20s → (C) 10s
Tsk3	(C) 10s → (E/S) 20s

9- Etude de cas (3)

Algorithme préemptif: quantum(tick)=10s et priorités différentes: on suppose que les commutations ne se font qu'en synchronisme avec les ticks (à l'inverse de freertos)

Running	2	2	2	i	1	2	3	1	1	i	1	1	1	i	
Ready	1	1	1	1	3	3	1	v	v	v	v	v	v	v	
	3	3	3	3		1									
Blocked	v	v	v	2	2	v	v	3	3	1					

B: Priorité forte → tâche plus prioritaire;

commutation de contexte synchronisée sur les ticks d'horloge

Tâche	Priorité	Comportement
Tsk1	2	(C) 30s→(E/S) 10s → (C) 25s
Tsk2	3	(C) 25s→(E/S) 15s → (C) 10s
Tsk3	2	(C) 10s→(E/S) 20s

Systèmes et réseaux temps réel (youssefrohdi@yahoo.fr)

51

9- Etude de cas (4)


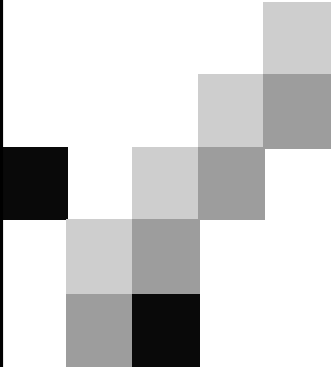
Algorithme non-préemptif: priorités égales ; quantum 5s

Running	1	1	1	1	1	1	2	2	2	2	2	3	3	1	1	1	1	2	2	i
Ready	2	2	2	2	2	2	3	3	3	3	3	1	1	v	2	2	2	2	v	v
	3	3	3	3	3	3			1	1	1									
Blocked	v	v	v	v	v	v	1	1	v	v	v	2	2	2	3	3	3	v	v	v
														3						

Tâche	Priorité	Comportement
Tsk1	2	(C) 30s→(E/S) 10s → (C) 25s
Tsk2	2	(C) 25s→(E/S) 15s → (C) 10s
Tsk3	2	(C) 10s→(E/S) 20s

Systèmes et réseaux temps réel (youssefrohdi@yahoo.fr)



52



Systèmes et réseaux temps réel
Chap III: Gestion des tâches

Systèmes et réseaux temps réel
(youssefrochdi@yahoo.fr)

53



Objectifs


- Apprendre à gérer sur freertos des tâches: créer, supprimer, suspendre, bloquer
- Illustrer le diagramme des états d'une tâche à travers des exemples sur l'implémentation de freertos sur simulateur.
- Réviser et corriger l'algorithme préemptif « simplifié » présenté précédemment concernant les événements asynchrones.
- Définir la tâche idle et illustrer son utilité.

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

54



1- Définition d'une tâche


- Une tâche est définie par une fonction classique **avec comme type de retour void, et ne contenant jamais de return.**

```
/*Définition typique*/
void ATaskFunction(void * pvParameters){
    /*pvParameters pointeur sur les arguments de la fonction*/
    /*variables locales*/
    int iVarLocal=0;
    /*une tâche est (en général) une boucle infinie*/
    for(;;){
        /*Travail effectué par la tâche a placer ici*/
    }
    /*si on atteint ce point la tâche doit être effacée*/
    vTaskdelete(NULL);
    /*jamais de return*/
}
```

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 55



2- Création d'une tâche (1)

- Une tâche est créée à l'aide de l'appel à la fonction de l'API:

```
portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
    /*pvTaskCode nom de la fonction définissant la tâche*/
    const signed portCHAR * const pcName,
    /*un nom attribué à cette tâche utile pour debuggage seulement*/
    unsigned portSHORT usStackDepth,
    /*profondeur de la pile pour cette tâche*/
    void *pvParameters,
    /*pointeur sur les paramètres à passer à la fonction tâche*/
    unsigned portBASE_TYPE uxPriority, /*priorité de tâche*/
    xTaskHandle *pxCreatedTask /*Handle désignant la tâche*/
);
```

- Une fois la tâche créée elle est dans un état Ready.

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 56

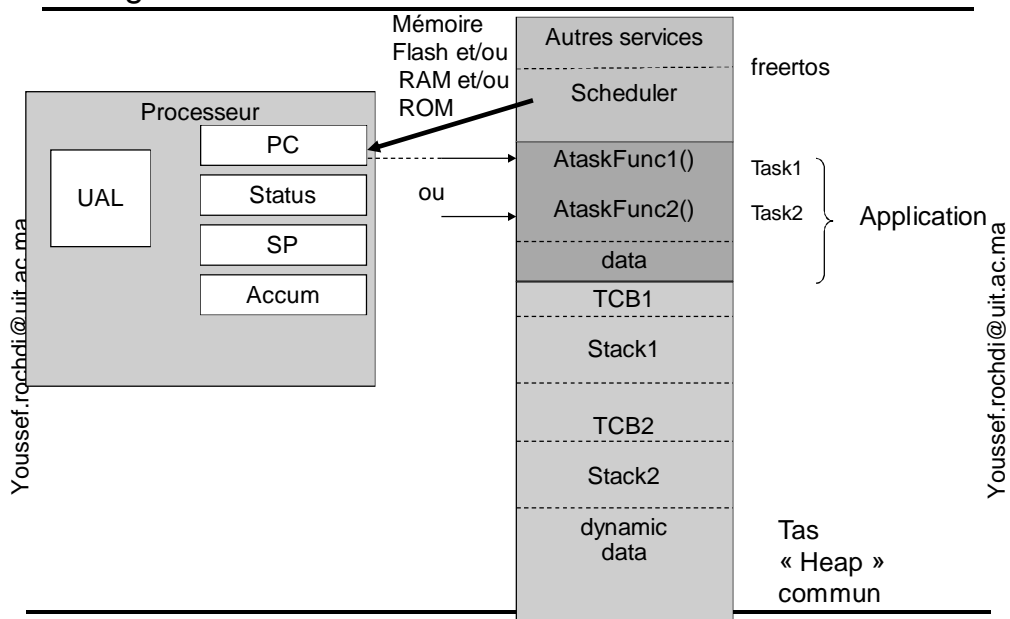
2- Création d'une tâche (2)

- Valeurs retournées par: `portBASE_TYPE xTaskCreate (...);`
 - `pdPASS` (équivalent `pdTRUE`, ou 1) si l'opération réussit.
 - `errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY` (équivalent -1 voir définition dans `projdefs.h`), si mémoire insuffisante.
- Freertos ne limite pas le nombre max de tâches, tout dépend de la mémoire disponible dans le système.
- Freertos alloue pour chaque tâche un bloc de contrôle de tâche (TCB) et une pile (Stack) dans la tas mémoire commun(Heap).

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

57

3- Organisation de la mémoire centrale



Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr)

58

freeRTOS

4- Suppression d'une tâche

- Une tâche est supprimée par appel à la fonction de l'API:

```
void vTaskDelete( xTaskHandle pxTask )
```

```
/* pxTask handle (identif) désignant la tâche à supprimer,
   si NULL la tâche à supprimer est la tâche appelante*/
```
- Elle sera supprimée de toute liste: Ready, suspended, blocked.
- Quand une tâche est supprimée c'est la tâche idle qui doit libérer l'espace mémoire {TCB + Stack} de la tâche supprimée.
- Idle TASK doit avoir la chance de s'exécuter pour faire le travail de nettoyage sinon risque de mémoire saturée.
- L'espace mémoire alloué dynamiquement doit être libéré par la tâche elle-même avant l'appel à vTaskDelete.

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 59

freeRTOS

5- Exploration des états Ready, Running(1)

Exemple1


- crée deux tâches qui sont deux boucles infinies, de même priorité 1 (Supérieure à celle de la tâche idle de priorité 0) et puis démarre le scheduler.
- La tâche Idle ne s'exécute jamais puisqu'il y a toujours une tâche plus prioritaire dans l'état Ready

Youssef.rochdi@uit.ac.ma

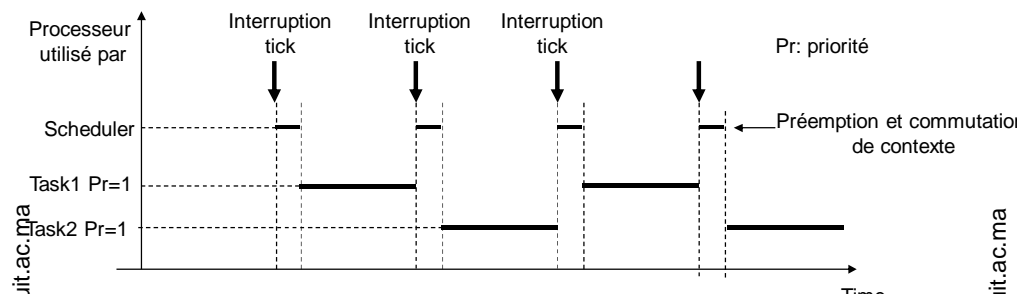
Youssef.rochdi@uit.ac.ma

S: Scheduler

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 60



5- Exploration des états Ready, Running (2)



Processeur utilisé par

Scheduler

Task1 Pr=1

Task2 Pr=1

Time

Interruption tick

Pr: priorité


Préemption et commutation de contexte

Fichier freertosconfig.h

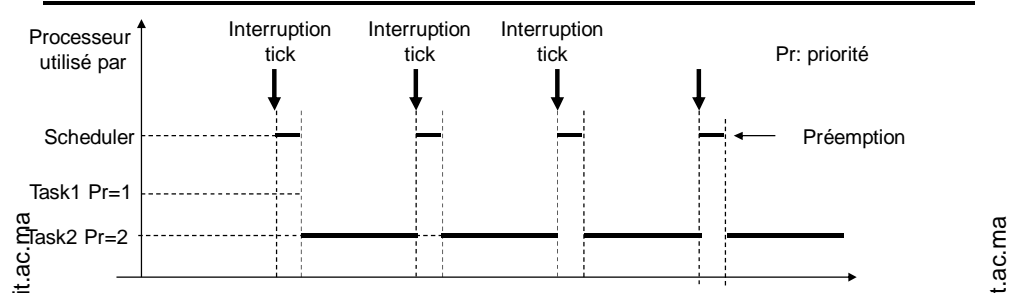
```

#define configUSE_PREEMPTION 1
#define configUSE_IDLE_HOOK 1
#define configUSE_TICK_HOOK 1
#define configTICK_RATE_HZ ( 1000 )
/* In this non-real time simulated environment
the tick frequency has to be at least a multiple
of the Win32 tick frequency,
and therefore very slow. */
#define configMINIMAL_STACK_SIZE ( ( unsigned short ) 50 )
/* In this simulated case, the stack only has to hold one
small structure as the real stack is part of the win32 thread. */
#define configTOTAL_HEAP_SIZE ( ( size_t ) ( 20 * 1024 ) )
    
```

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 61



5- Exploration des états Ready, Running (3)



Processeur utilisé par

Scheduler

Task1 Pr=1

Task2 Pr=2

Time

Interruption tick

Pr: priorité

Préemption

Fichier main.c

```

/* Create one of tasks Here. */
/*--->To complete.....*/
xTaskCreate(vTask1,"Task1",1000,NULL,1,&xTaskHandle1);
xTaskCreate(vTask2,"Task2",1000,NULL,2,&xTaskHandle2);
    
```

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 62

freeRTOS

5- Exploration des états Ready, Running (4)

Pr: priorité
Préemption et comm contexte

Scheduler
Task1 Pr=0
Task2 Pr=0
idle Pr=0

Fichier main.c

```

/* Create one of tasks Here. */
/*--->To complete....*/
xTaskCreate(vTask1,"Task1",1000,NULL,0,&xTaskHandle1);
xTaskCreate(vTask2,"Task2",1000,NULL,0,&xTaskHandle2);
    
```

Time

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 63

freeRTOS

6- priorités dynamiques


- Les priorités des tâches peuvent être lues ou modifiées à l'aide de:


```

unsigned portBASE_TYPE uxTaskPriorityGet( xTaskHandle
pxTask );

void vTaskPrioritySet(xTaskHandle pxTask, unsigned
portBASE_TYPE uxNewPriority);
            
```
- Les priorités des tâches sont définies entre 0 (idle priority) à configMAX_PRIORITIES-1 (définie dans freertosconfig.h)
- En principe, la priorité maximale n'est limitée que par le type portBASE_TYPE, mais plus cette priorité maximale est petite plus freertos consomme moins de mémoire: freertos place les tâches prêtes dans différentes files selon leur priorités (de 0 jusqu'à configMax_Priorities-1)

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 64




6- priorités dynamiques

```
void vTaskUrgence( void * pvParameters ){
    TRISA=0;
    while(1) {
        if (_RB0==1) //Arrêt d'urgence appuyé
        { vTaskPrioritySet(TskUrg_handle, uxTaskPriorityGet(TskUrg_handle)+3);
          declenche_sirene();
          while(_RB0==1){//tant que AU appuyé clignoter PORTA
            PORTA = 0x00; delay();//delai software
            PORTA = 0xFF; delay(); }
          arret_sirene(); PORTA=0x00;
          vTaskPrioritySet(TskUrg_handle, uxTaskPriorityGet(TskUrg_handle)-3);
        } } }
```

Yo Yo

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 65



7- Exploration de l'état bloqué (1)

- Une tâche est dans un état bloqué si elle ne s'exécute pas et attend un événement (interne: fin de temporisation par exemple ou externe: arrivé d'un caractère sur USART).
- Quand l'événement apparait la tâche passe à l'état Ready et le scheduler est alors appelé pour choisir une tâche à exécuter.
- Mais en fait, la fin d'une temporisation coïncide avec un interrupt-tick
- Une tâche quitte l'état Ready vers l'état Blocked si elle a fait appel à une fonction bloquante.
- Avantage: meilleure utilisation du processeur

Youssef.rochdi@uit.ac.ma Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 66

freeRTOS

7- Exploration de l'état bloqué (2)

- Un exemple de fonction bloquante est


```
void vTaskDelay( portTickType xTicksToDelay );
```

`xTicksToDelay` indique le nombre d'interruption-ticks à attendre.
Exemple: compteur de ticks=1000, puis appel à `vTaskDelay(20)`, déblocage quand compteur tick=1020
- Le nombre d'interruption à attendre est convertible en ms:


```
// Blocks for 500ms.
const portTickType xDelay = 500 / portTICK_RATE_MS;
vTaskDelay(xDelay );
```

Exemple 2:
Crée 3 tâches task1(Pr=1), Task2 (Pr=1), Task3 (Pr=2)
Task3 est une boucle infinie, se bloque pendant 4 ticks après avoir fait un traitement.

67

freeRTOS

7- Exploration de l'état bloqué (3)

Pr: priorité

Scheduler

Task3 Pr=2

Task2 Pr=1

Task1 Pr=1

Time

Interruption tick

T

Préemption et comm contexte


≈4 Ticks

Traitement précédent appelé À `vTaskDelay()`

Task3 se bloque `Count_Tick=X` Asynchrone

Task3 se débloque et préempte Task1 `Count_Tick=X+4` Synchrones

68



7- Exploration de l'état bloqué (4)


- `vTaskDelay(portTickType xTicksToDelay)`
Fixe un délai d'attente par rapport à l'instant de son appel (Relatif), si cet instant varie à cause du code qui le précède, il est difficile de produire une tâche périodique avec une période précise.
- Freertos propose pour des tâches périodiques avec une période fixe, la fonction bloquante:

```
void vTaskDelayUntil( portTickType *pxPreviousWakeTime,  
    portTickType xTimeIncrement );
```

`pxPreviousWakeTime`: dernier instant absolu de déblocage, mis à jour automatiquement et doit être initialisé au premier appel

`xTimeIncrement`: prochain déblocage à l'instant absolu:
`(pxPreviousWakeTime + xTimeIncrement)`

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 69




7- Exploration de l'état bloqué (5)

- Utilisation: tâche de périodicité 100 Ticks

```
// Perform an action every 100 ticks.  
  
void vTaskFunction( void * pvParameters ) {  
    portTickType xLastWakeTime;  
    const portTickType xTicks = 100;  
  
    // Initialise the xLastWakeTime variable with the current time.  
    xLastWakeTime = xTaskGetTickCount ();  
    for( ;; ) {  
        // Wait for the next cycle.  
        vTaskDelayUntil( &xLastWakeTime, xTicks );  
        // Perform action here.  
    }  
}
```

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 70




7- Exploration de l'état Suspendu

- Une tâche est dans un état suspendu est une tâche dormante, elle ne s'exécute pas, elle n'est pas prête pour s'exécuter et n'est pas donc prise en compte par le scheduler et elle n'attend aucun événement.
- Le seul moyen de la faire sortir de cet état est un appel explicite à la fonction `vTaskResume()`.
- Une tâche rentre dans un état suspendu par un appel explicite à la fonction `vTaskSuspend()`.
- Une tâche peut suspendre les autres tâches pour garder le monopole du processeur jusqu'à ce qu'elle termine un certain traitement en appelant `vTaskSuspendAll()`.
- `vTaskSuspendAll()` suspend les activités du kernel freertos (swapping entre autres), mais pas les interruptions et le comptage de ticks.
- Pour reprendre ces activités on appelle `xTaskResumeAll()`.

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 71



7- Exploration de l'état Suspendu

- `xTaskResumeAll()` retourne `pdTRUE (1)` si une commutation de contexte est nécessaire, `pdFALSE (0)` sinon.
- NB: L'appel à `xTaskResumeAll()` doit correspondre à un appel à `vTaskSuspendAll()`, sinon il y aura des erreurs de fonctionnement: il doit y avoir autant d'appel à `xTaskResumeAll()` que d'appel à `vTaskSuspendAll()` pour reprendre les activités du scheduler.

Youssef.rochdi@uit.ac.ma

Youssef.rochdi@uit.ac.ma

Systèmes et réseaux temps réel (youssefrochdi@yahoo.fr) 72

