وزارة البحث العلمي والتعليم العالي
**MINISTERE DE L'ENSEIGNEMENT SUPEREUR ET DE LA RECHERCHE SCIENTIFIQUE**
جامعــة عبد الحميد بن باديس مستغانم
**Université Abdelhamid Ibn Badis Mostaganem**
كـلـيـة الـعــلــوم والـتـكـنـولـوجـيـا
**Faculté des Sciences et de la Technologie**
**DEPARTEMENT DE GENIE ELECTRIQUE**

**N° d'ordre : M ....../GE/2020**

# MEMOIRE

Présenté pour obtenir le diplôme de

## MASTER EN ELECTRONIQUE DES SYSTEME EMBARQUES

Par

**Larbi Abdelkader**

# ETUDE ET ANALYSE FONCTIONNELLE DE FreeRTOS

Encadré par : **Mr Ould Mammar Madani**

Soutenu le devant le jury composé de :

| | |
|---|---|
| Président : | Université de Mostaganem |
| Examinateur : | Université de Mostaganem |
| Examinateur : | Université de Mostaganem |

Année Universitaire 2019/2020

# Remerciements

*La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude.*

*Je voudrais tout d'abord adresser toute ma reconnaissance à mon encadreur Monsieur Ould Mammar M., pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.*

*Je désire aussi remercier tout le personnel de l'Université Abdelhamid Ibn Badis de Mostaganem en particulier celui du département de génie électrique de la faculté des sciences et technologies.*

*Enfin, la liste ne pouvant pas être exhaustive, permettez-moi d'adresser mes sincères remerciement à ma petite et grande familles, amis, proches et toute personne qui mon encouragé et cru en moi dès le début.*

## Résumé:

Les systèmes temps réel sont de plus en plus omniprésents avec la prolifération des systèmes embarques, plus petits, plus puissants et peu couteux. Ces systèmes sont utilisés dans des domaines très variés, et font partie de notre vie de tous les jours. Ceci a rendu l'électronique de plus en plus présente sous plusieurs variétés de forme : les moyens de transport, les téléphones portables, le multimédia, et tout système embarqué.

Les systèmes embarqués deviennent de plus en plus complexes. C'est pour cette raison que les concepteurs ont recours à l'emploi d'un noyau temps réel multitâche (RTOS) pour gérer cette complexité.

Ce RTOS permet à une application d'être visualisée sous forme d'un ensemble de modules qui s'exécutent quasi-simultanément sur un même processeur. Le système temps réel doit également pouvoir prendre immédiatement en compte les stimuli synchrones provenant du matériel, il doit donc pouvoir gérer les interruptions. On associe classiquement à chaque interruption, un module appelé routine d'interruption (ISR), qui réalise l'interface entre l'application et son environnement matériel. Ces routines d'interruption, encore appelés séquences immédiates, ont pour but de répercuter l'occurrence d'un évènement *(seuil de température franchi, capteur activé, etc. ...)* à la tâche chargée de son traitement.

FreeRTOS est un système d'exploitation temps réel gratuit et (open source) développé par

***Real Time Engineers***, parmi ses fonctionnalités figurent les caractéristiques suivantes : des tâches préemptives, un support de plusieurs architectures de microcontrôleurs, un faible encombrement (4,3Ko sur un ARM7 après compilation3), écrit en C et compilé avec divers compilateurs . Il permet également un nombre illimité de tâches à exécuter en même temps et aucune limitation quant à leurs priorités tant que le matériel utilisé peut se le permettre.

Enfin, il implémente des files d'attente, des sémaphores binaires et de comptage et des mutex.



## ملخص:

أصبحت أنظمة الوقت الحقيقي أكثر انتشارًا مع انتشار الأنظمة الموجودة على متن الطائرة ، وأصغر حجمًا وأكثر قوة وأقل تكلفة. تُستخدم هذه الأنظمة في مجموعة متنوعة من المجالات ، وهي جزء من حياتنا اليومية. وقد أدى هذا إلى ظهور الإلكترونيات بشكل متزايد في العديد من الأشكال: وسائل النقل ، والهواتف المحمولة ، والوسائط المتعددة ، وأي نظام على متن الطائرة.

أصبحت الأنظمة المدمجة أكثر تعقيدًا. لهذا السبب يلجأ المصممون إلى استخدام نواة متعددة المهام في الوقت الحقيقي (RTOS) للتعامل مع هذا التعقيد.

يتيح نظام RTOS عرض التطبيق على أنه مجموعة من الوحدات التي تعمل في وقت واحد تقريبًا على معالج واحد. يجب أن يكون نظام الوقت الحقيقي أيضًا قادرًا على أن يأخذ في الاعتبار المحفزات المتزامنة من الأجهزة على الفور ، لذلك يجب أن يكون قادرًا على إدارة الانقطاعات. تقليديًا ، ترتبط كل مقاطعة بوحدة نمطية تسمى روتين المقاطعة (ISR) ، والتي توفر الواجهة بين التطبيق وبيئة الأجهزة الخاصة به. تهدف إجراءات المقاطعة هذه ، والتي تسمى أيضًا التسلسلات الفورية ، إلى تمرير وقوع حدث (عبر عتبة درجة الحرارة ، وتنشيط المستشعر ، وما إلى ذلك) إلى المهمة المسؤولة عن معالجتها.

FreeRTOS هو نظام تشغيل فوري (مفتوح المصدر) مجاني تم تطويره بواسطةReal Time Engineers
من بين وظائفه الخصائص التالية: المهام الاستباقية ، ودعم العديد من هياكل المتحكم الدقيقة ، والبصمة الصغيرة (4.3 كيلو بايت على ARM7 بعد التجميع 3) ، ومكتوبة بلغة C ومجمعة مع مجمعين مختلفين. كما يسمح بتنفيذ عدد غير محدود من المهام في نفس الوقت ولا توجد قيود على أولوياتهم طالما أن الأجهزة المستخدمة قادرة على تحمل ذلك.
أخيرًا ، فإنه ينفذ قوائم الانتظار ، والإشارات الثنائية والعد والعلامات.

**Abstract:**

Real-time systems are more and more ubiquitous with the proliferation of smaller, more powerful and inexpensive on-board systems. These systems are used in a wide variety of fields, and are part of our everyday life. This has made electronics increasingly present in many varieties of forms: means of transport, cell phones, multimedia, and any on-board system.

Embedded systems are becoming more and more complex. It is for this reason that designer's resort to using a real-time multitasking kernel (RTOS) to handle this complexity.

This RTOS allows an application to be viewed as a set of modules that run almost simultaneously on a single processor. The real-time system must also be able to immediately take into account synchronous stimuli from the hardware, so it must be able to manage interruptions. Classically, each interrupt is associated with a module called the interrupt routine (ISR), which provides the interface between the application and its hardware environment. These interrupt routines, also called immediate sequences, are intended to pass on the occurrence of an event (temperature threshold crossed, sensor activated, etc.) to the task responsible for its processing.

FreeRTOS is a free and (open source) real-time operating system developed by
Real Time Engineers, among its features are the following characteristics: preemptible tasks, support for several microcontroller architectures, small footprint (4.3KB on an ARM7 after compilation3), written in C and compiled with various compilers. It also allows an unlimited number of tasks to be performed at the same time and no limitation on their priorities as long as the hardware used can afford it.

Finally, it implements queues, binary and counting semaphores, and mutexes.

# Sommaire

***Résumé*** ........................................................................................................3

***Introduction générale*** ........................................................................................10

***I Chapitre 1 : ECLIPSE C/C++*** ...........................................................................13

I.1 Introduction ........................................................................................................ 14

I.2 Installer et Configurer Eclipse .......................................................................... 14

I.2.1. Installer Eclipse à l'aide du package dédié au développement C/C++ ....... 14

I.2.2. Ajouter le plug-in CDT à une installation d'Eclipse existante .................. 15

I.3. Installation de l'outillage C/C++ sous Windows ............................................... 16

I.3.1. Installation de MinGW ................................................................................ 17

I.3.2. Installation de Cygwin ................................................................................. 17

I.4. Vérification du fonctionnement dans Eclipse .................................................... 18

I.4.1. Création d'un Projet C/C++ ......................................................................... 18

I.4.2. Exécution en mode Debug ........................................................................... 21

I.4.3. Ajuster le paramètre d'un Projet C/C++ ...................................................... 24

I.5. Conclusion .......................................................................................................... 27

***II Chapitre 2 : Débogage d'applications embarquées*** ..........................................28

II.1 Conception et outils de débogage ..................................................................... 29

II.1.1 Introduction ................................................................................................. 29

II.1.2 Conception de débogage ............................................................................. 30

II.1.3 Techniques de débogage ............................................................................. 31

II.1.4 Mode de débogage ...................................................................................... 31

II.1.4.1. Stop mode ................................................................................................31

II.1.4.2. Run mode .................................................................................................31

II.1.5. RTOS aware débogueur ............................................................................. 32

II.1.5.1. Les probepoints ........................................................................................32

II.1.5.2. ReTIS ........................................................................................................34

II.1.5.3. Trace utilisée ............................................................................................36

II.1.6. Conclusion ....................................................................................................37

***III Chapitre 3 : FreeRTOS Analyse Fonctionnelle*** ...............................................38

III.1 Aperçu sur FreeRTOS ...................................................................................... 39

III.1.1 Caractéristiques générales .......................................................................... 39

III.1.2 Distribution du code source ........ 41

III.2 Description du Noyau FreeRTOS ........ 42

III.2.1 Introduction ........ 42

III.2.2 Configuration de FreeRTOS ........ 42

III.2.3 Gestionnaire de Taches ........ 44

III.2.3.1. Bloc de contrôle d'une tache ........ 44

III.2.3.2. Diagramme d'état d'une tache ........ 45

III.2.4 Gestion des listes ........ 46

III.2.4.1. Listes « Ready » et « Blocked » ........ 47

III.2.4.2. Initialisation des listes ........ 48

III.2.4.3. Insertion dans une liste ........ 49

III.2.5. L'Ordonnanceur de FreeRTOS(FreeRTOS Scheduler) ........ 53

III.2.5.1. Le cadre du contexte de la tache ........ 55

III.2.5.2. Démarrage et arrêt des taches ........ 56

III.2.5.3. Commutation entre les taches ........ 60

III.2.5.4. Démarrage du Scheduler ........ 60

III.2.5.5. Suspension du Scheduler ........ 61

III.2.5.6. Vérification de la liste des taches retardées (Delayed Task List) ........ 65

III.2.5.7. Traitement de la section critique (Critical Section Processing) ........ 65

***IV. Chapitre 4 : FreeRTOS+TRACE ........ 66***

IV.1. FreeRTOS + Trace ........ 67

IV.1.1. Introduction ........ 67

IV.1.2. Présentation du système ........ 68

IV.1.3. Utilisation FreeRTOS+Trace ........ 69

IV.2. Enregistreur Bibliothèque ........ 74

IV.2.1. Utilisation de l'enregistreur de trace ........ 74

IV.2.1.1. Hardware Timer Porting ........ 75

IV.2.2. FreeRTOS intégration ........ 75

IV.2.3. Télécharger les données de trace ........ 76

IV.3. Fenêtre principale ........ 76

IV.3.1. Trace View ........ 76

IV.3.1.1. Mode d'affichage ........ 77

IV.3.1.2. Evénements ........................................................................ 78
IV.3.1.3. Zoom et les fonction de navigation ................................................ 78
IV.3.1.4. Outil de tableau ........................................................................ 79
IV.3.1.5. Les fonction avancées ................................................................ 79
IV.3.2. Les options de menu .......................................................................... 80
IV.3.2.1. Fichier ...................................................................................... 80
IV.3.2.2. Trouver .................................................................................... 80
IV.3.2.3. voir .......................................................................................... 81
IV.4. Fenêtre du Finder ................................................................................. 81
IV.5. Horizontale Trace View ......................................................................... 82
IV.6. Charge CPU Graphique ........................................................................ 83
IV.7. Kernel utilisation d'objets ..................................................................... 85
IV.8. Intensité de planification ...................................................................... 85
IV.9. Service de noyau Appel ........................................................................ 86
IV.10. Temps Bloquer noyau Appel ................................................................ 86
IV.11. Utilisation Signal Plot événement ......................................................... 87
IV.12. Acteur instance Graphique .................................................................. 88
IV.13. Débit de comunication ........................................................................ 90
IV.13.1. Appliquer des filtres ......................................................................... 92
IV.14. Rapport de Statistique ........................................................................ 92
IV.15. Journal des événements utilisateur ....................................................... 94
IV.16. Historique de l'objet du noyau .............................................................. 95
***V. Chapitre 5 : Application*** ......................................................................... ***96***
V.1. Application FreeRTOS + Trace et FreeRTOS + CLI ....................................... 97
V.1.1. Utilisation de la FreeRTOS Win32 Simulator ............................................. 97
V.1.1.1. Présentation ............................................................................... 97
V.1.1.2. Fonctionnalité ............................................................................. 98
V.1.2. Construire et exécuter l'application ......................................................... 99
V.2. Création d'un enregistrement Trace FreeRTOS+ ......................................... 104
V.2.1. Voir l'enregistrement Trace Trace dans FreeRTOS+ ................................... 105
***Conclusion générale*** ............................................................................. ***108***
***Bibliographie*** ....................................................................................... ***112***

# Liste des figures

Figure I-1 : Fenêtre de contrôle de préférences dans Eclipse ........ 15
Figure I-2 : menu de sélection ........ 16
Figure I-3 : Fenêtre d'assistant pour création d'un projet C/C++ ........ 19
Figure I-4 : Fenêtre du code source du fichier principale ........ 20
Figure I-5 : Code source ........ 21
Figure I-6 : Fenêtre représentant l'application compilée ........ 22
Figure I-7 : Fenêtre Debug Configurations ........ 23
Figure I-8 : Fenêtre Propities for MonProgramme C++........ 24
Figure I-9 : Fenêtre Propities for MonProgramme C++ avec la variable PATH ........ 25
Figure I-10 : Fenêtre Propities au niveau du menu Tool Chain Editor ........ 25
Figure I-11 : Fenêtre Propities au niveau de l'onglet Tool Settings ........ 26
Figure II-1 : Principe du fonctionnement des Probepoints........ 33
Figure II-2 : Environnement de ReTiS ........ 35
Figure II-3 : Trace sur la d'emo 186 AMD........ 36
Figure III-1 : FreeRTOS Source Distribution ........ 41
Figure III-2 : FreeRTOSConfig.h (PIC18) ........ 42
Figure III-3 : TCB de FreeRTOS ........ 45
Figure III-4 : Diagramme d'état d'une tâche dans FreeRTOS ........ 45
Figure III-5 : Listes Créées par le Scheduler ........ 47
Figure III-6 : Type xList........ 47
Figure III-7 : Types xList ........ 48
Figure III-8 : Initialisation de liste........ 48
Figure III-9 : ***vlistInsert*** avec Arguments ........ 49
Figure III-10 : DelayedTaskList........ 50
Figure III-11 : Code Extrait de List.c dans FreeRTOS ........ 51
Figure III-12 : Quelle liste à utiliser ? (de **Tasks.c**) ........ 52
Figure III-13 : Inverser les pointeurs de listes lors d'un débordement du timer........ 53
Figure III-14 : Scheduler Algorithm........ 54
Figure III-15 : Stacking of MCU context ........ 55
Figure III-16 : Vue de Création d'une tâche ........ 57

Figure III-17 : Allocate Stack and TCB Memory ............................................................ 58

Figure III-18 : Dummy Stack Frame .................................................................................. 59

Figure III-19 : FreeRTOS Task Scheduler Startup............................................................ 60

Figure III-20 : Algorithms for ***vTaskSuspond*** and ***xTaskResumeAll***................................ 62

Figure III-21 : Algorithm for ***vTaskIncrementTick***.......................................................... 64

Figure V-1 : Les paramètres du terminal requis YAT...................................................... 104

Figure V-2 : Capture d'écran après la trace RTOS a été démarré et arrêté ...................... 105

Figure V-3 : Le RTOS enregistré trace dans FreeRTOS+Trace ..................................... 106

# *Introduction générale*

Il existe un grand nombre de RTOS (Systèmes d'exploitation Temps réel), libres, *Open Sources* et propriétaires (RTX, MicroC/OS III, FreeRTOS, VxWorks...) possédant des jeux d'avantages et d'inconvénients différents afin de s'adapter au très grand nombre de problématiques du marché. Nous avons porté notre attention sur FreeRTOS, un *scheduler* temps réel offrant un certain nombre d'avantages. La documentation de FreeRTOS est directement accessible en ligne depuis le site internet ( http://www.freertos.org/ ).

En 2014, FreeRTOS est l'un des RTOS leader sur le marché de l'embarqué. FreeRTOS est libre de droit d'utilisation et open source (sous licence). Il est officiellement supporté par une trentaine d'architectures (TI, Microchip, Atmel, NXP, Intel ...) et 18 chaînes de compilation. Il a par exemple été téléchargé plus de 100000 fois l'année passée, cela implique une communauté assez riche d'utilisateurs. Il existe de plus en tout 4 variantes de FreeRTOS :

- FreeRTOS : basique
- FreeRTOS + Trace : idem FreeRTOS avec des outils propriétaires permettant d'instrumenter le code. Par exemple des outils graphiques de trace.
- OpenRTOS : version commerciale sous licence de FreeRTOS
- SafeRTOS : version certifié SIL3 TUV De plus en 2016, FreeRTOS est le RTOS Actuellement le plus utilisé et celui le plus regardé pour démarrer de nouveaux projets.

Observons le marché en 2015 des OS et RTOS pour l'embarquer (www.eetimes.com) :

FreeRTOS, comme n'importe quel autre noyau, est un outil purement logiciel, ce n'est qu'un système de fichiers. FreeRTOS nous propose une API (Interface de Programmation d'Applications) de programmation pour gérer un environnement multitâche.

Nous voulons par ce travail , montrer la puissance de ce noyau et ça facilitée d'implémentation dans plusieurs axes en occurrence dans le domaine pédagogique en l'utilisant comme outil de formation au sein des institutions et écoles spécialisées afin d'étudier les systèmes temps réels vu que FreeRTOS est gratuit, accessible et surtout très bien documenté.

Nous aborderons dans ce travail les principales caractéristiques et fonctionnalités du noyau FreeRTOS par :

- Eclipse c/c++ (IDE).
- Débogage d'application embarquées.
- L'analyse fonctionnelle de FreeRTOS.
- FreeRTOS+Trace.
- APPLICATION.

# *I Chapitre 1 : ECLIPSE C/C++*

## I.1. Introduction

Eclipse est un IDE, Integrated Development Environment (EDI environnement de développement intégré). Il est développé par IBM, est gratuit et disponible pour la plupart des systèmes d'exploitation.

Ce chapitre a pour objectif de mettre en place un environnement de développement C/C++ avec Eclipse.

Le développement en C/C++ avec Eclipse requiert le plug-in officiel CDT, ainsi qu'un compilateur C/C++, l'outil make et un débuggeur.

## I.2. Installer et Configurer Eclipse

En premier lieu, il faut rassembler les éléments nécessaires en ce qui concerne Eclipse.

Deux choix possibles :

1 - Partir de zéro et installer le package Eclipse, préassemblé avec le plug-in CDT.

2 - Ajouter le plug-in CDT à une installation d'Eclipse existante.

Quelle que soit l'option choisie, vous disposerez à l'issue des mêmes composants vous permettant de développer en C/C++ sous Eclipse.

### I.2.1. Installer Eclipse à l'aide du package dédié au développement C/C++

Eclipse est un EDI qui permet l'ajout de toutes sortes de plug-ins, notamment le plug-in CDT, spécifiquement conçu pour le développement C/C++. Ainsi, si on dispose déjà d'une installation d'Eclipse ne comportant pas encore le plug-in CDT, l'ajouter sera aisé et suffirait pour développer en C/C++.

Pour autant, faire le choix de partir de zéro en installant le package Eclipse préassemblé pour le C/C++, permet de commencer sur des bases saines et dans des conditions optimales.

Eclipse IDE for C/C++ Developers, disponible sur cette page http://www.eclipse.org/downloads/.

L'installation de celui-ci se résume ensuite à décompresser le fichier récupéré, à un endroit approprié, avec comme seul prérequis, le fait de disposer d'une machine virtuelle Java, sans laquelle Eclipse ne pourra pas démarrer.

### I.2.2. Ajouter le plug-in CDT à une installation d'Eclipse existante

Si on dispose d'une installation d'Eclipse et pour laquelle on ignore si elle est déjà équipée du plugin CDT, un moyen rapide de le vérifier consiste à contrôler la présence de la rubrique C/C++ au niveau du menu Windows > Préférences dans Eclipse.

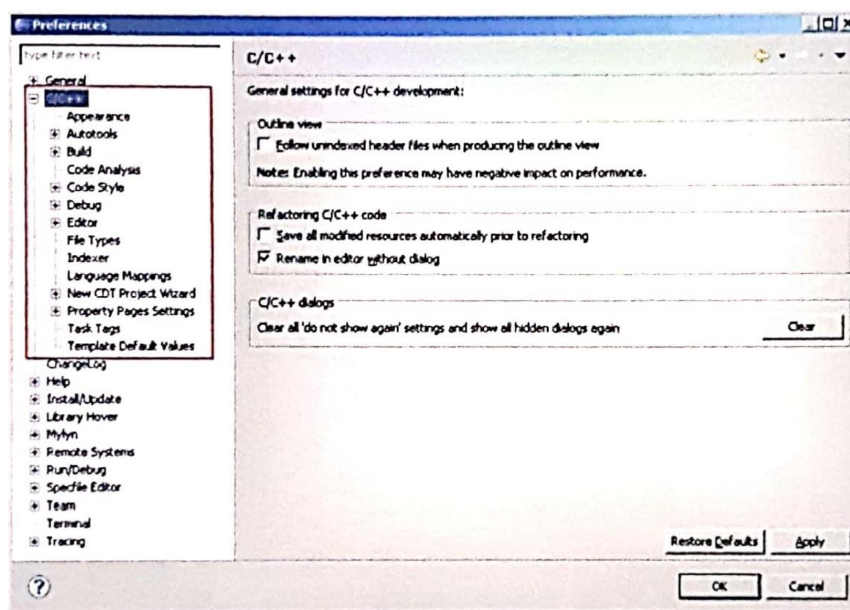


Figure I-1 :Fenêtre de contrôle de préférences dans Eclipse

Si on dispose d'une installation d'Eclipse qui ne contient pas encore le plug-in CDT, alors il faut suivre les étapes suivantes :

1 - Allez dans le menu Help > Install New Software...

2 - Dans la liste Work with, sélectionnez l'update-site correspondant à votre version, c'est-à-dire : (Eclipse Version - http://download.eclipse.org//releases/ec/ipseversionl

Par exemple, pour Eclipse Juno (4.2), on aura l'update-site suivant :
(Juno - http://download.eclipse.org/releases/junol

3 - Une fois le bon update-site sélectionné, vous devez cocher les plug-ins concernant le C/C++.

Au minimum, prenez soin de sélectionner ceux-ci :

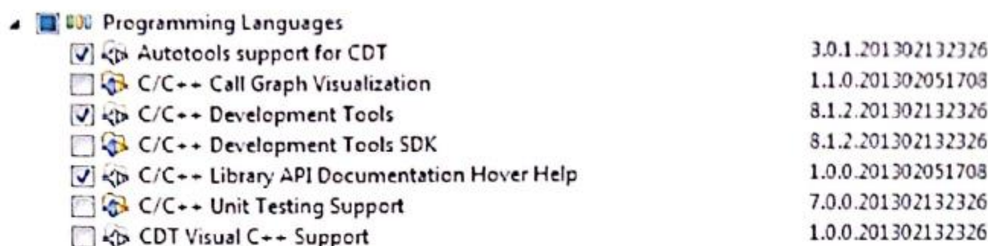


Figure I-2 : menu de sélection

4 - On Clique sur Finish puis en suit les instructions jusqu'au redémarrage d'Eclipse.

Une fois Eclipse redémarré, nous pouvons d'ores et déjà créer des projets C/C++ et ouvrir les fichiers source dans l'éditeur de code C/C++. Cependant, si le système ne comporte pas encore de compilateur, outil make et débugger, on ne peut pas compiler et exécuter les programmes. Le cas échéant, on doit installer ces outils, parallèlement à Eclipse, comme indiqué ultérieurement.

## I.3. Installation de l'outillage C/C++ sous Windows

Eclipse et son plug-in CDT n'étant pas suffisants pour développer des programmes en C/C++, il convient de compléter ceux-ci avec les outils permettant de produire des programmes exécutables.

Pour cela, il y a notamment MinGW et Cygwin, lesquels rassemblent les éléments permettant d'obtenir une chaine de compilation cohérente et complète. Ceux-ci embarquent tous deux le package binutils lequel contient entre autres l'éditeur de liens Id, le générateur d'archives ran, le générateur d'index pour les bibliothèques ranlib..., mais aussi la suite GCC avec différents compilateurs tels que C, C++, Fortran, ADA... et leurs bibliothèques.

De plus, MinGW et Cygwin sont également pourvus de tout un système d'émulation de Linux sous Windows. Ceci permet par exemple de profiter des Autotools, autorisant ainsi le lancement de scripts de configuration, dans le but de compiler des outils supplémentaires.

A noter que MinGW et Cygwin ne sont pas les seules collections d'outils disponibles mais on se focalisera principalement sur celles-ci.

Le choix de l'utilisation de MinGW ou de Cygwin doit se faire en fonction du type d'exécutable que l'on souhaite produire.

**Remarque :** On peut tes installer parallèlement, sans que cela ne pose de problème. Eclipse saura déterminer l'ensemble des outils dont il peut disposer, à partir du moment où ceux-ci auront été référencés dans la variable d'environnement PATH du système ou dans la configuration d'un projet.

### I.3.1. Installation de MinGW

MinGW étant la contraction de "Minimalist GNU for Windows", cela signifie que c'est un environnement sous licence GNU, dédié au développement d'applications natives pour Windows. Il ne requiert pas d'autre bibliothèque que celles fournies par le système Windows mais ne fournit pas d'environnement d'exécution POSIX.

MinGW apporte les éléments suivants :

- Un portage de la collection d'outils de compilation GNU (GCC), incluant les compilateurs C, C++, ADA et Fortran.
- GNU Blnutils pour Windows (assembleur, linker, gestionnaire d'archives).
- Un installateur en ligne de commande (mlngw-get) pour le déploiement de MinGW et MSYS sur Windows.
- Une IHM graphique (mingw-get-lnst) permettant d'installer MinGW et les composants souhaités.

Pour finaliser l'installation de MinGW, il reste à ajouter à la variable d'environnement PATH, le chemin vers le répertoire bin de MinGW.

Pour vérifier la bonne prise en compte de votre paramétrage, ouvrez une nouvelle Invite de commande DOS (via Démarrer > Exécuter... > cmd.exe) et tapez la commande set. Ceci fait alors apparaître la liste des variables d'environnement, au sein desquelles sera présente la variable Path, complétée du chemin que vous venez d'ajouter.

**Remarque :** Si Eclipse est démarré, le relancer afin d'être sûre qu'il puisse prendre en compte les modifications de la variable PATH.

### I.3.2. Installation de Cygwin

Comme expliqué précédemment, Cygwin est composé d'une collection d'outils qui servent à émuler le "Look and Feer• Linux, sous Windows. Autrement dit, Cygwin permet de retrouver l'apparence et les usages de l'environnement Linux, à l'aide des mêmes programmes et utilitaires présents sur ce système.

D'autre part, a contrario de MinGW, Cygwin permet de créer des exécutables au standard POSIX, donc il vous revient de savoir si cela vous convient car dans ce cas, vos applications devront pouvoir disposer de l'environnement POSIX pour s'exécuter.

Pour installer Cygwin, on procède comme suit :

1 - Télécharger la dernière version de Cygwin à cette adresse http://cygwin.com/setup.exe, puis l'exécuter. Lorsque ta première fenêtre s'ouvre, on sélectionne les options qui conviennent et cela, jusqu'au formulaire de sélection des packages.

2 - Par défaut, un certain nombre de packages sont présélectionnés mais en ce qui concerne les outils C/C++, ce n'est pas le cas. De fait, il faut rechercher et cocher un à un les packages, pour le compilateur, l'outil make et le débugger.

Tous les packages qui nous intéressent se trouvent dans la rubrique **Develop**.

Pour finaliser l'installation de Cygwin, il reste à ajouter le chemin vers son répertoire bin, à la variable d'environnement **PATH**.

Validez en cliquant sur **Ok**, jusqu'à sortir des **Propriétés Système**.

Pour vérifier la bonne prise en compte de votre paramétrage, ouvrez une nouvelle **Invite de commande DOS** (via **Démarrer** > **Exécuter**... > **cmd.exe**) et tapez la commande set. Ceci fait alors apparaître la liste des variables d'environnement, au sein desquelles sera présente la variable **Path**, complétée du chemin.

## I.4. Vérification du fonctionnement dans Eclipse

Cygwin et MinGW sont dédiés aux environnements Windows, donc une fois ceux-ci installés convenablement, Eclipse dispose en principe du nécessaire pour compiler ou débugger un programme C/C++. Sur Linux ou Mac OS X, Eclipse devrait pouvoir en faire de même, à partir du moment où les conditions évoquées en introduction sont réunies.

Pour vérifier qu'Eclipse est 100% opérationnel pour développer en C/C++, nous allons créer un projet de type Hello World, le compiler, l'exécuter, puis nous passerons au mode Debug.

Si ce n'est déjà fait, lancez Eclipse et placez-vous dans la perspective C/C++, afin notamment de pouvoir accéder facilement aux outils dédiés à ce type de développement. Si besoin, cette perspective peut être activée en la sélectionnant par l'intermédiaire de son icône, lequel est par défaut placé en haut à droite.

### I.4.1. Création d'un Projet C/C++

Voici les étapes pour créer un projet :
1 - Allez dans le menu File > New > C++ Project.

Dans l'assistant qui s'affiche, commencer par donner un nom au projet, par exemple MonPrgmC++.

Au niveau du cadre Project Type, sélectionner Executable > Hello World C++ Project.

Puis dans le cadre Toolchains, sélectionnez Cygwin GCC ou MinGW GCC selon les choix précédents, puis cliquez sur Next > jusqu'au dernier formulaire ou cliquez directement sur Finish.

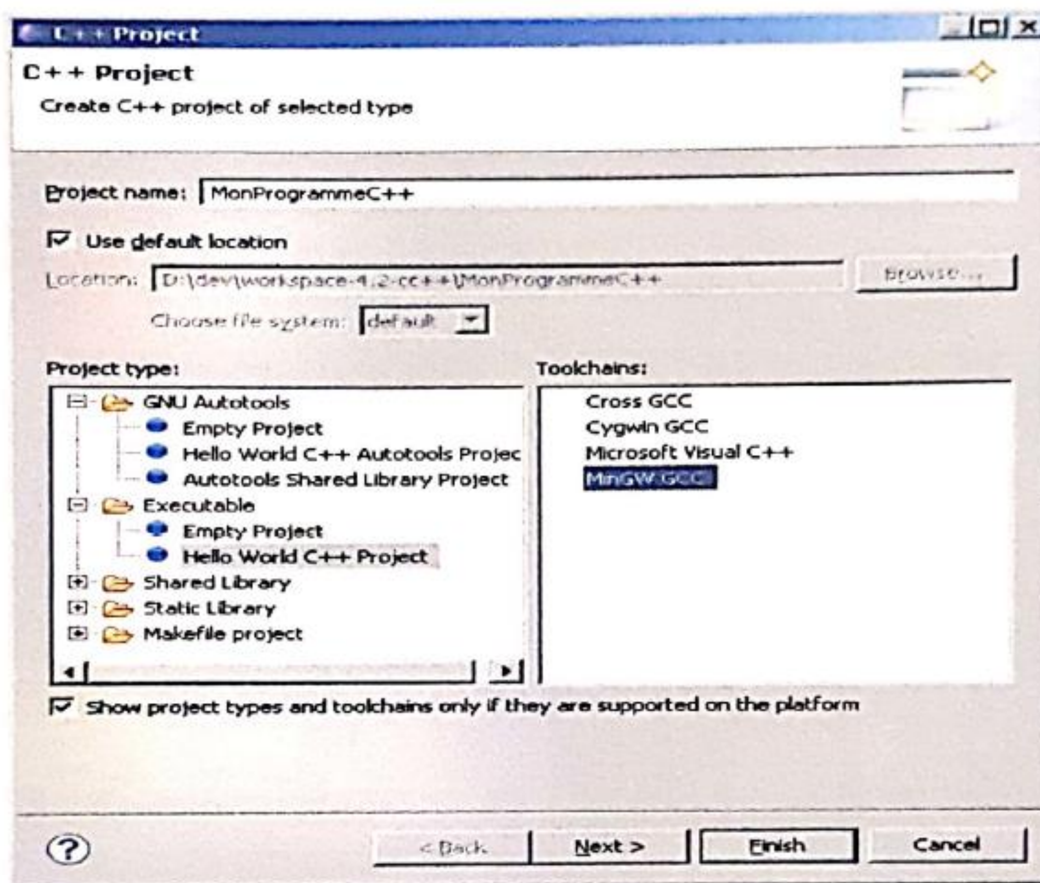



Figure I-3 : Fenêtre d’assistant pour création d’un projet C/C++

2 - Le projet est en place et on devra voir le code source du fichier principal.

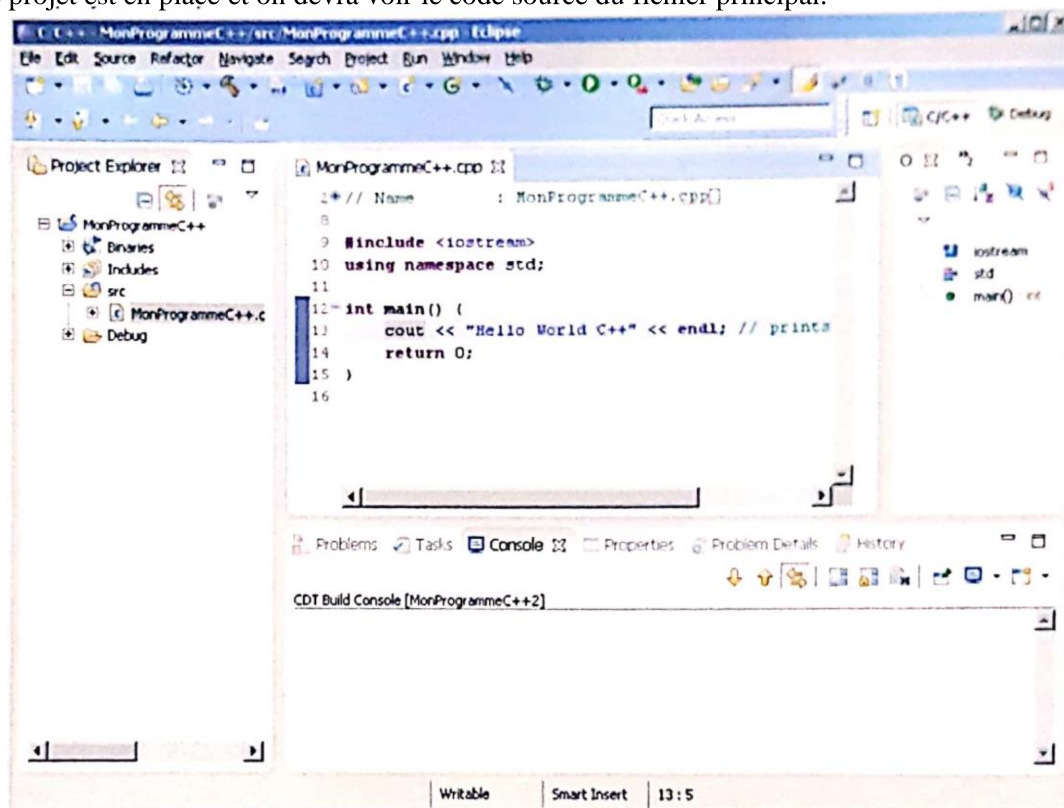


Figure I-4 : Fenêtre du code source du fichier principale

3 - Sélectionner le fichier source du projet, puis on lance la compilation en allant dans le menu **Project** > **Build Project.**

20:52:50 **** Rebuild of configuration Debug for project MonProgrammeC++ ****

Info: Internal Builder is used for build

g++ -00 -g3 -Wall -c -fmessage-length=0 -o "src\\MonProgrammeC++.o""..\\src\\MonProgrammeC+

g++ -o MonProgrammeC++.exe "src\\MonProgrammeC++.o"

20:52:52 Build Finished (took Is.532ms)

4 - Pour finir, un clic droit sur le fichier source, ou sur le projet puis Run As > Local C/C++ Application, ce qui provoque l'affichage de la chaîne attendue, dans la console : Hello World C++

## I.4.2. Exécution en mode Debug

Pour exécuter l'application en mode Debug, voici comment procéder :

1- Pour rendre le test plus explicite, on va ajouter une ligne simple de code supplémentaire. Par exemple, ajoutez la ligne qui suit, juste avant le **return.**

cout << "Et ça continue! " << endl;

2- Dans la marge à gauche de la ligne que vous venez d'ajouter, double-cliquez afin de faire apparaître un point bleu représentant un point d'arrêt. Vous pouvez faire la même opération via le clic droit, toujours dans la marge gauche, puis en sélectionnant **Toggle Breakpoint.**

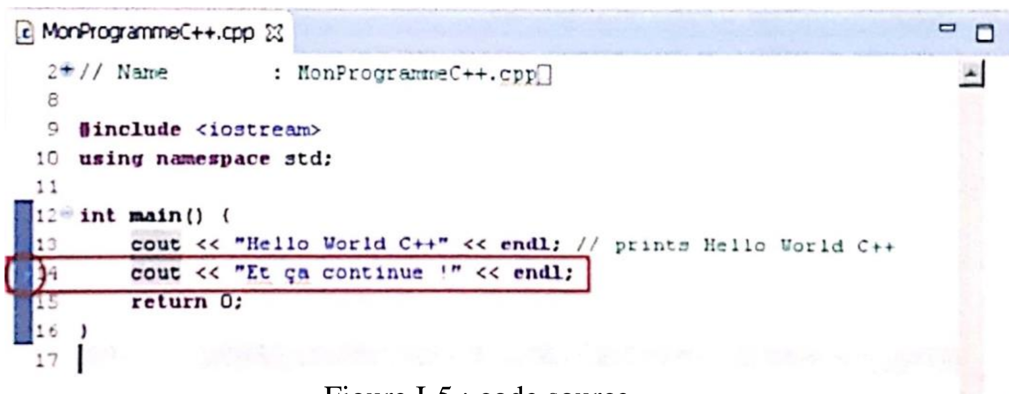



Figure I-5 : code source

3- On exécute à présent l'application en mode Debug, en allant dans le menu **Run** > **Debug** ou en appuyant sur **F11**.

On constate que l'application est automatiquement compilée avant de s'exécuter.

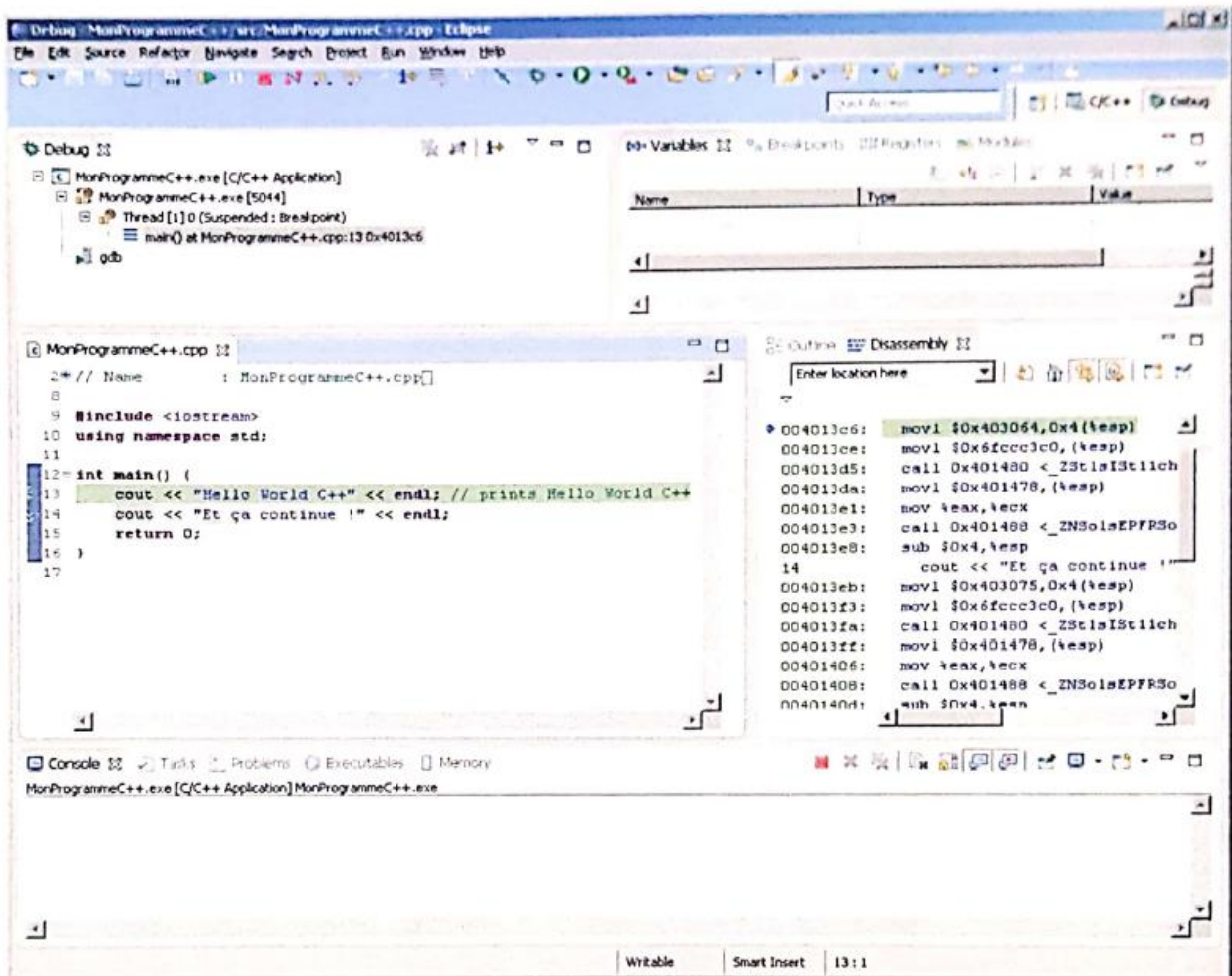


Figure I-6 : Fenêtre représentant l'application compilée

4- Le mode Debug s'arrête sur la première ligne du corps de la méthode main et non pas sur notre point d'arrêt.

En fait, cela est paramétrable au niveau du menu Run > Debug Configurations, dans l'onglet Debugger de la configuration d'exécution de l' application. Ainsi, sur cette page, on peut configurer le mode Debug afin qu'il s'arrête ou non sur la méthode indiquée.

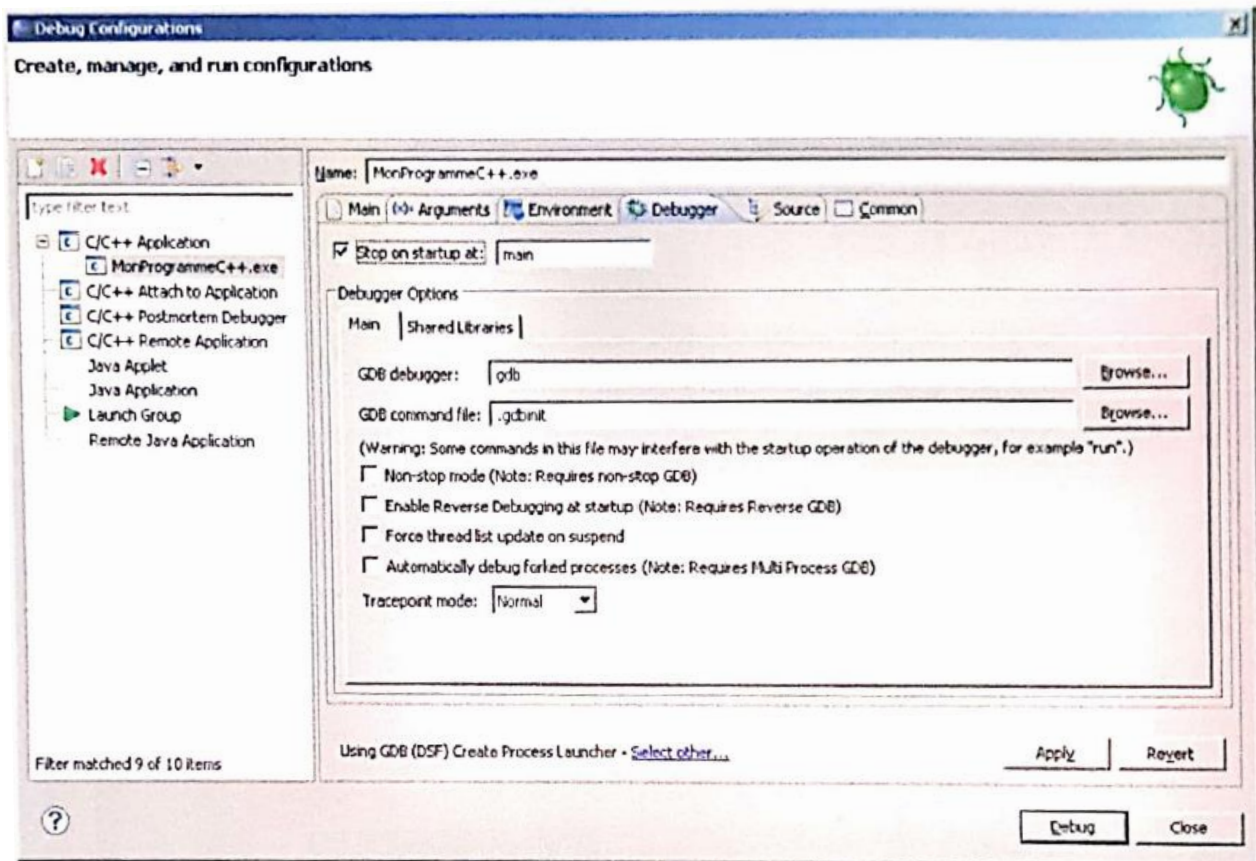



Figure I-7 : Fenêtre Debug Configurations

5 - Si on appuie sur F8, le mode Debug continue son exécution et s'arrête sur le premier point d'arrêt suivant rencontré, donc dans le cas présent, sur celui qu'on a ajouté.

Pour relancer le déroulement, il suffit de nouveau d'appuyer sur la touche F8.

En résumé les touches utiles pour le mode Debug sont les suivantes :

- F5 : Rentre dans la méthode invoquée sur la ligne courante.
- F6 : Exécute l'instruction courante sans entrer dans la méthode invoquée.
- F7 : Sort de la méthode courante.
- F8 : Continue l'exécution jusqu'au prochain Point d'Arrêt ou la fin du programme.

Le mode Debug est extrêmement utile et permet d'agir de façon précise sur l'exécution.

## I.4.3. Ajuster le paramètrage d'un Projet C/C++

Eclipse offre de nombreuses possibilités de configuration au sein d'un projet C/C++.

Il est possible de modifier l'outillage C/C++ d'un projet, c'est-à-dire de passer par exemple de Cygwin à MinGW, ou inversement. Cette opération permet d'avoir un premier aperçu de ce qui caracterise un projet C++ au sein d'Eclipse

Ainsi, pour faire basculer l'outillage C/C++ de MinGW vers Cygwin ou inversement, voici comment procéder :

1 - Un clic droit sur le projet dans la vue Project Explorer, puis aller dans le menu

Properties > C/C++ Build > Environment.

Si on a choisi l'outillage de MinGW, on doit obtenir la fenêtre suivante avec la variable PATH qui commence par le chemin vers les exécutables de MinGW :

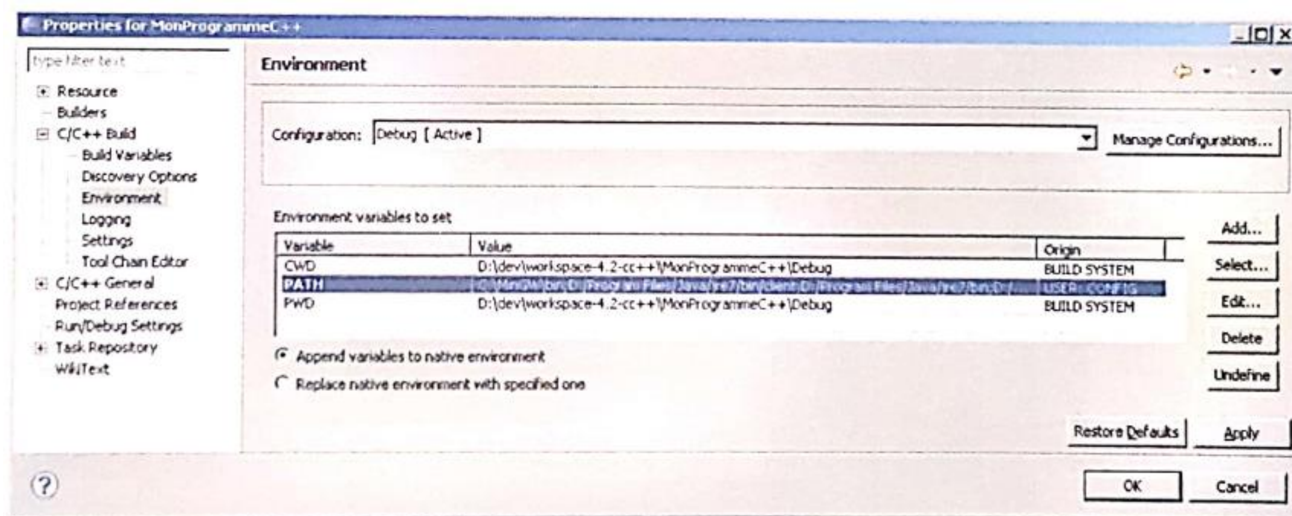



Figure I-8 : Fenêtre Propities for MonProgramme C++

Si en revanche on a choisi l'outillage de Cygwin, on doit obtenir la fenêtre suivante avec la variable PATH qui commence par le chemin vers les exécutables de Cygwin :

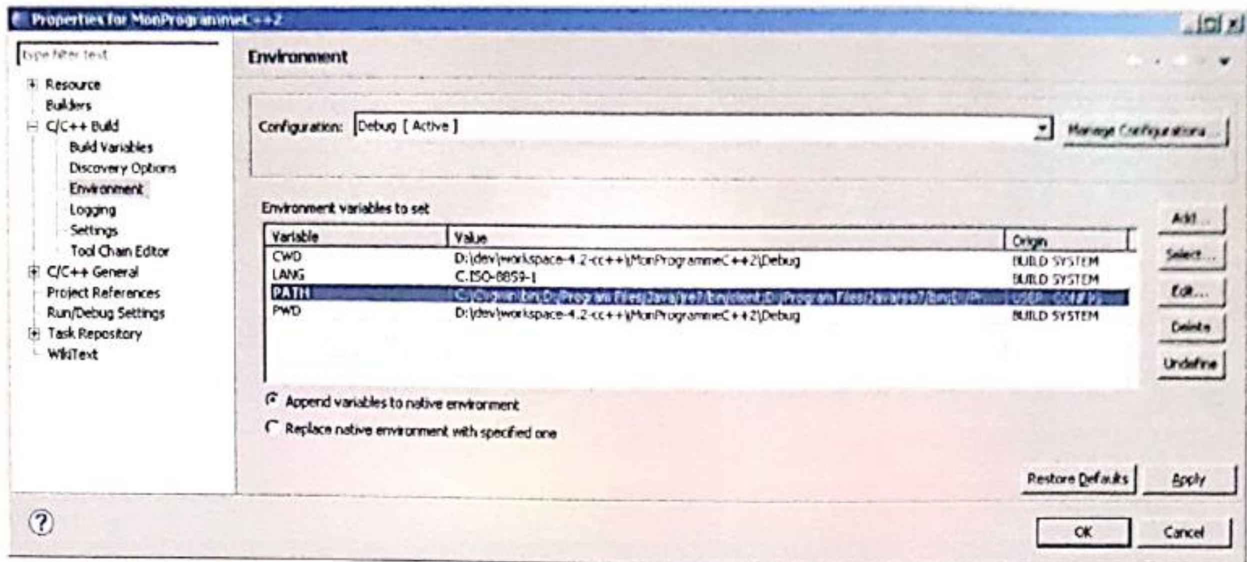



Figure I-9 : Fenêtre Propities for MonProgramme C++ avec la variable PATH

On Note que les valeurs indiquées peuvent varier en fonction de note environnement et de nos choix.

2 - On peut modifier la variable PATH à l'aide du bouton Edit..., en indiquant le chemin vers l'outillage qui convient, l'important étant que celui-ci soit placé en premier dans cette variable. On peut choisir de supprimer ou laisser figurer le chemin de l'ancien outillage dans le PATH, du moment qu'il n'est pas en premier.

3 - Pour que le changement d'outillage soit complet, il reste à l'indiquer explicitement, au niveau du menu Tool Chain Editor. Ainsi, il faut modifier le Current toolchain en conformité avec la modification de la variable PATH.

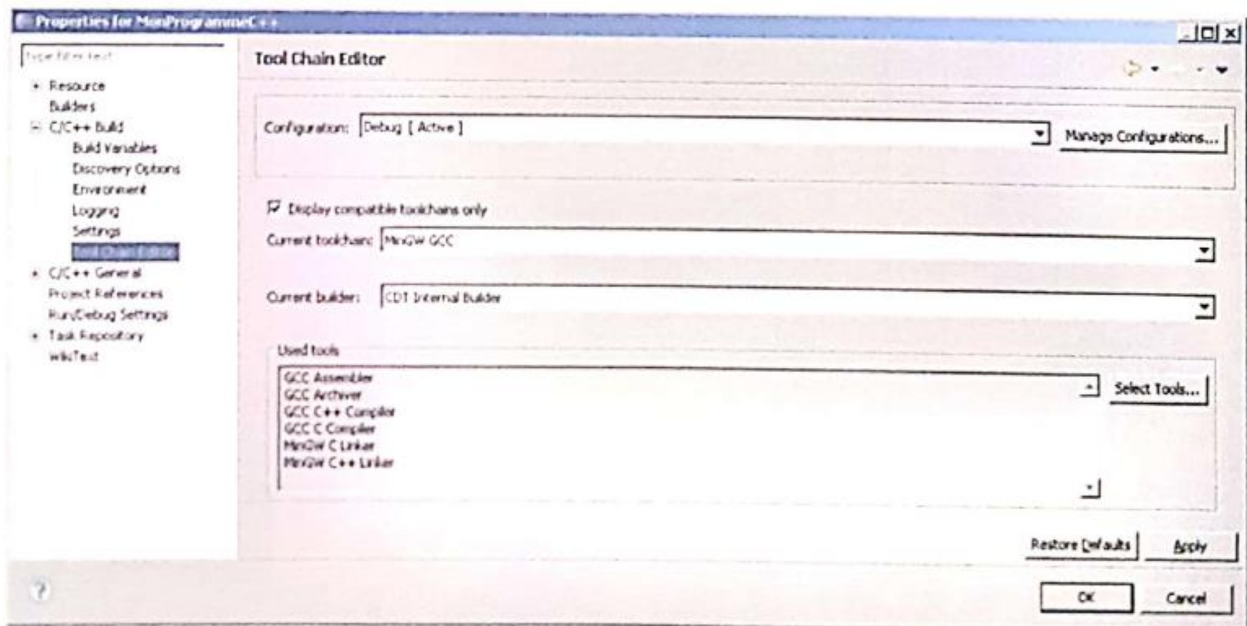



Figure I-10 : Fenêtre Propities au niveau du menu Tool Chain Editor

4 - A ce niveau-là, la modification de l'outillage est terminée, il ne reste plus qu'à cliquer sur Ok pour valider la modification. Toutefois, pour qu'il soit facile de vérifier que tout est bien pris en compte, on peut ajouter une option qui permettra de visualiser les versions et emplacements des outils utilisés au moment de la compilation.

Pour cela, aller dans le menu Settings, puis dans l'onglet Tool Settings, sélectionner par exemple GCC C++ Compiler > Miscellanous, et enfin cochez l'option Verbose (-v) comme indiqué ci-dessous :

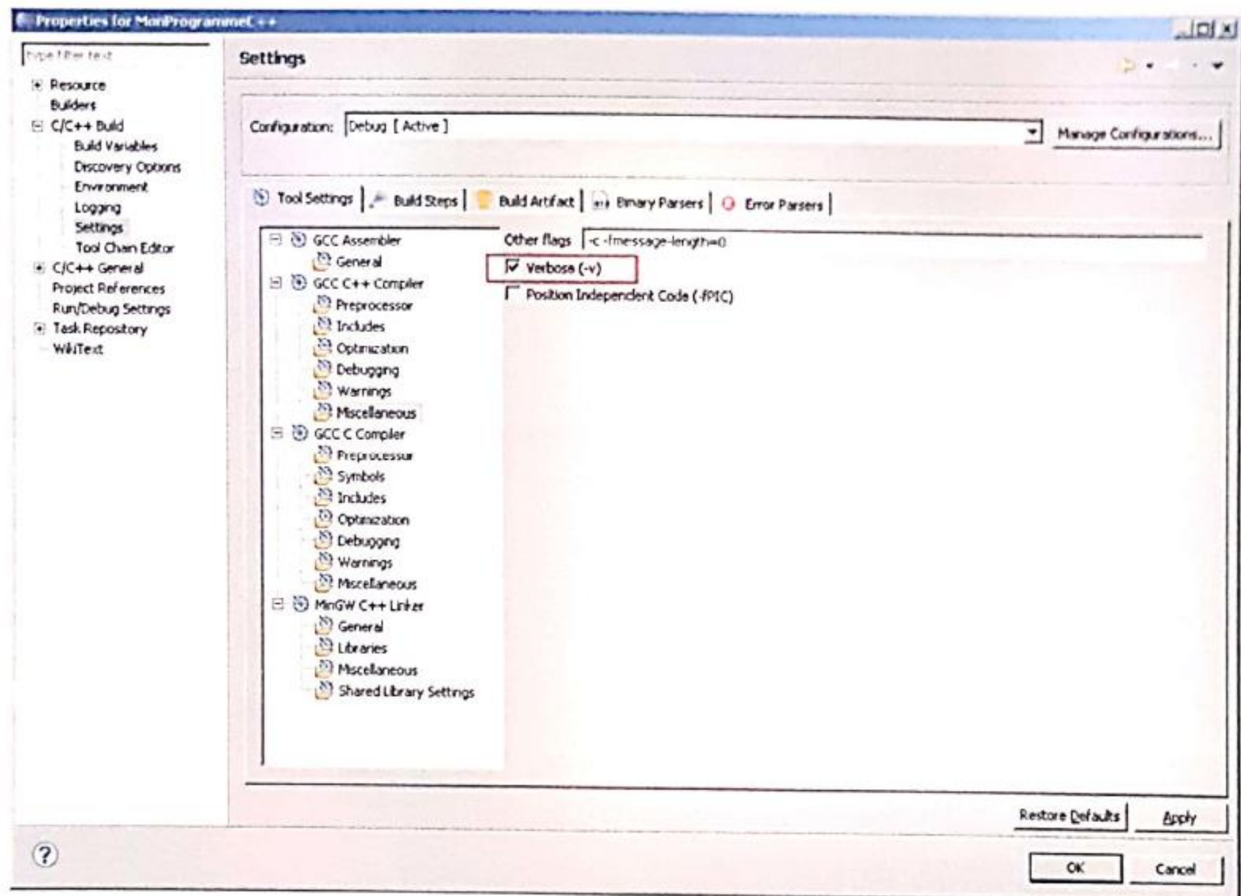



Figure I-11 : Fenêtre Propities au niveau de l'onglet Tool Settings

Pour terminer cliquez sur **Ok**, sélectionnez votre projet, puis effectuez un **Project > Clean** suivi d'un **Project > Build**.

Ainsi, dans la console d'Eclipse, on devra voir apparaître un certain nombre d'informations supplémentaires, (le chemin du compilateur, sa version...).

## I.5. Conclusion

Ainsi on peut démarrer le développement d'une application C/C++ avec Eclipse.

Nous avons vu qu'Eclipse a besoin d'outils externes de façon à pouvoir compiler et exécuter des programmes C/C+ mais il est important de savoir qu'ils existent d'autres outils en dehors de MinGW et Cygwin, tels que SDK Windows, MinGW-w64, TDM-GCC...

Les versions successives d'Eclipse apportent leur lot de nouveautés et on peut dire que le projet CDT est en constante évolution. Il en est à sa version 8.1.2 (Année 2013) et d'autres évolutions le concernant sont au programme. Notamment, il est prévu qu'Eclipse intègre les particularités de la version 11 du langage C++.

# *II Chapitre 2 : Débogage d’applications embarquées*

Dans cette partie, nous recensons les différents outils de débogage et par la suite, nous chercherons à déterminer les caractéristiques essentielles que doit posséder un outil de vérification de système temps réel embarqué.

Nous verrons dans un premier temps le principe général d'un débogueur de type RIOS aware et par la suite nous présentons quelques types de débogueur temps réel tels que les probepoints et ReTiS, ensuite la trace des appels système.

## II.1. Concepts et outils de débogage

La première démarche effectuée pour réaliser nos outils de débogage temps réel, fut d'étudier les solutions actuellement proposées.

Ce chapitre passe brièvement en revue les différents axes de recherche pour le débogage des applications temps réel. Ces méthodes de débogage ne prennent pas en compte les mêmes phénomènes, et sont utilisées à différents stades selon le domaine de développement.

### II.1.1. Introduction

Comme les applications embarquées deviennent plus complexes, les développeurs résultent de plus en plus des systèmes d'exploitation en temps réel (RTOS) pour manipuler cette complexité. Cependant, de nouveaux défis apparaissent quand il s'agit de faire le débogage des applications temps réel.

Il faut noter que, de toutes les activités d'un ingénieur concepteur, celle du débogage prend le plus de temps et risque d'annuler l'avantage primaire de l'utilisation d'un RTOS, dans le cas où ce temps n'est pas maîtrisable. En effet, le débogage d'un système embarqué temps réel peut s'avérer être une tâche très ardue.

Plusieurs aspects liés aux applications multitâches ne sont pas pris en charges par les outils classiques de débogage.

Typiquement, la plupart de ces outils de débogage du code source qu'on peut classer comme étant des outils à bas niveau peuvent seulement afficher des informations telles que des variables locales ou celles des registres de la machine cible, ou des appels liés à la pile relative à l'état de la tâche en cours ou à son contexte, c'est-à-dire la tâche actuellement en exécution. Il n'y a pas de moyen pour visualiser en même temps les autres tâches. De même, une fonction réentrante peut être appelée par plusieurs tâches simultanément, donc mettre un point d'arrêt (breakpolnt) dans cette fonction pour déboguer une tâche bien déterminée, peut être inutile pour faire le débogage d'une autre tâche. Pour résumer, on peut citer les quelques problèmes pertinents à une application multitâche et qui sont les problèmes [10] :

— d'exclusion mutuelle,

— de verrou mortel (deadlock),

— d'affectation de priorité,

— d'inversion de priorité

— de débordement de pile, etc...

De plus, en utilisant seulement les informations fournies par un débogueur classique pour suivre l'interaction de multiples tâches, ainsi que leur synchronisation par sémaphores ou par évènements, et les communications inter tâches par passation de messages, imposerait la connaissance détaillée du fonctionnement interne du RTOS ainsi que sa structure de données. Au cas Su le code source n'est pas disponible, ceci rendra la tâche de débogage plus difficile et consommera beaucoup de temps.

Un autre défi à relever est celui de déboguer une application multitâche qui soit embarquée sur une machine cible de type microcontrôleur par exemple. La raison est due au fait qu'il est difficile de déboguer le code sur la cible alors que l'application est en exécution, en d'autres termes déboguer en temps réel.

### II.1.2. Concepts de débogage

La seule conventionnelle et viable configuration pour le débogage d'un système embarqué à base de noyau temps réel est de connecter une machine host (PC exécutant le débogueur) à la machine cible. La cible peut être le système embarqué ou peut être une simulation qui s'exécute aussi sur la machine host.

Généralement, deux types de connexion host/cible peuvent être utilisés pour le débogage :

— Connexion dédiée au débogage - ou la liaison n'a pas de fonctions additionnelles à part le débogage

— Ligne de communication - qui peut être utilisée pour d'autres buts, en plus du débogage.

La connexion dédiée au débogage devient de plus en plus utilisée. Typiquement, elle est basée sur le standard JTAG (Joint Test Action Group). Avant JTAG, Motorola avait introduit le BDM (Background Debug Mode), qui fonctionne d'une manière similaire. Presque toutes les nouvelles conceptions de processeurs incorporent l'interface JTAG pour le besoin de débogage. Un nombre réduit de broches du processeur est utilisé pour une communication série et synchrone avec le poste host.

Historiquement. L'utilisation d'une ligne de communication pour le débogage était très répandue. Généralement c'est une liaison série (RS232, RS422), ou Ethemet ou USB.

### II.1.3. Techniques de débogage

On peut classifier en deux catégories, les techniques de débogage des applications multitâches temps réelles ; celle des outils classiques et celle des outils orientés RTOS (RTOS aware debugger).

La technique des outils classiques ne prend pas en charge les aspects multitâches temps réel. En ce qui concerne les débogueurs orientés RTOS, deux modes de fonctionnement existent comme décrits ci-dessous.

### II.1.4. Modes de débogage

Il existe deux types de débogages de système temps réel, le "Stop Mode" et le "Run Mode" .Le Stop Mode est presque toujours une option, la disponibilité du Run Mode est avantageuse.

#### II.1.4.1. Stop mode

Aussi connu sous le nom de "Freeze Mode", ce mode existe quand tout le système entier est stoppé ou arrêté, quand aucun code n'est en exécution. Ce mode a lieu quand un point d'arrêt "breakpoint" est exécuté ou bien quand l'opérateur intervient. Ce mode réside sur la machine host, et peut 'être suffisant pour la majorité des situations, mais il est sans aucune utilité dans les cas :

— Le bogue recherché est étroitement lié avec l'interaction dynamique entre les tâches.
— Stopper le système entièrement, peut engendrer des effets secondaires qui peuvent gêner la location du bogue.
— Les évènements liés aux interruptions doivent être pris en charge.

Le JTAG utilise ce mode le processeur cible est stoppé d'exécuter le code de l'application pendant qu'il communique avec la machine host. Son inconvénient majeur est qu'il n'est pas disponible sur tous les processeurs.

#### II.1.4.2. Run mode

C'est un mode de d'débogage en exécution qui représente une technique beaucoup plus sophistiquée.
II permet d'arrêter une partie de système, par exemple :

— Seulement la tâche courante s'arrête (les autres continuent leur exécution).

—Seulement un groupe de Caches s'arrête (les autres continuent leur exécution).
—Toutes les Caches 'arrêtent, les interruptions sont prises en compte mais la tâche courante continue son exécution.

## II.1.5. RTOS aware débogueur

Ce qui est nécessaire pour déboguer, d'une manière efficace, une application utilisant un RTOS (ou RT-Kernel), ce sont des outils qui soient dédiés pour le suivi et l'analyse des aspects multitâche et temps réel, en d'autres termes des outils de type "RTOS awareness”. En effet,

bien que les outils classiques de débogage, tels les débogueurs au niveau source et les outils de profilage ou trace fournissent des informations utiles au sujet d'un système, ils ne fournissent seulement qu'une image statique d'une situation dynamique.

Donc, il est clair que pour déboguer un système embarqué utilisant un noyau temps réel, l'outil utilisé doit posséder deux caractéristiques fondamentales. La première a trait à l'aspect multitâche temps réel -caractéristique RTOS aware- et la deuxième doit permettre le débogage de l'application pendant son exécution et sur la machine.

De plus, il existe l'option ou le débogueur peut exercer une certaine forme de contrôle sur le système embarqué. Quelques possibilités de contrôle sur les tâches peuvent être implémentées telles que :

— Une tâche peut être suspendue

— Une tâche peut être retardée, pendant que les autres continuent leur exécution

— La priorité d'une tâche peut être ajustée, etc...

Dans la section suivante nous donnons une description de quelques outils qui permettant de déboguer une application temps réel embarquée. Parmi ces outils il y a Probepoints, ReTiS, Trace hook, ce sont des instruments qui nous aident à déboguer le noyau temps réel embarqué et fournir des renseignements déboguant utiles sur l'état du noyau et ses différents objets.

### II.1.5.1. Les probepoints

**Introduction**

Un Probe point est une ligne du programme ou vous allez brancher une sonde (en anglais probe).

Lors de l'exécution du programme, chaque fois que l'on rencontre un Probe point, le système va lire des valeurs dans un fichier sur le PC et les transférer dans la mémoire du système embarqué ou bien écrire dans un fichier le contenu de la mémoire du système embarqué. C'est très utile pour tester des appels systèmes à partir de données parfaitement définies dans un fichier qui peut par ailleurs être utilisé sous Matlab, Builder C++ ou dans tout autre logiciel. L'outil CrossView utilise cette technique.

Le principe du débogage standard sous CrossView est le suivant : soit l'utilisateur envoie un signal d'interruption (bouton HALT), soit l'application passe sur un breakpoint. Dans les deux cas, le moniteur prend le contrôle du microcontrôleur, interrompant l'exécution du programme en cours. S'en suit alors un dialogue entre CrossView et le moniteur permettant d'explorer l’état de

l'application (valeur des variables, de la mémoire).

Pendant toute la durée de ce dialogue, le moniteur garde le contrôle exclusif du microcontrôleur. Il ne le rend que lorsque l'utilisateur redémarre manuellement l'application (bouton GO).

Si cette solution est acceptable dans de nombreuses applications, il n'en va pas de même pour les applications basées sur un OS temps réel (comme le contrôle de moteurs...). Alors pour ces applications il faudrait un débogage qui serait lui aussi temps réel. Par débogage temps réel, il est possible de connaître l'état de l'application sans être obligé de l'arrêter, ou du moins en la perturbant le moins possible.

Pour résoudre ce problème, ils ont réfléchi à la solution suivante : établir un dialogue direct entre le noyau temps réel et CrossView, sans passer par le moniteur.

Du côté de la carte cible, le problème est relativement simple. Il suffira d'implémenter une tâche dédiée au débogage, une sorte de moniteur "temps réel". Cette tâche sera la plupart du temps en "sommeil" (elle ne prendra pas le contrôle du microcontrôleur) mais restera à l'écoute d'un signal extérieur. A la réception, de ce signal, la tâche récoltera les informations nécessaires sur le RTOS et les enverra au débogueur. Si la priorité de cette tâche est forte, ces opérations peuvent être effectuées sans occasionner de gêne trop importante aux autres tâches.

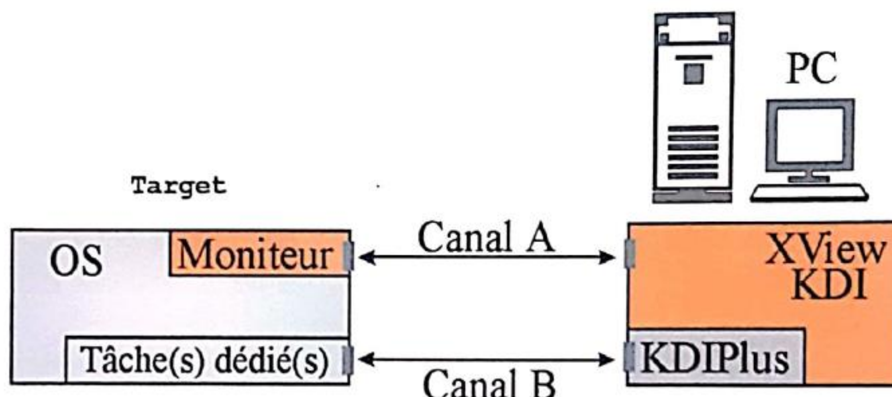



Figure II-1 : Principe du fonctionnement des Probepoints.

Du coté PC, le KDlPlus sera chargé d'envoyer des messages vers la tâche de débogage, mais aussi d'être à l'écoute des messages envoyés par cette dernière.

**Principe général de fonctionnement**

Le principe de fonctionnement des probe points est le suivant : via KDlPlus, l'utilisateur peut envoyer divers messages vers une tâche de noyau temps réel chargée de débogage.

Ces messages peuvent être scindés en deux catégories :

— Les messages de configuration.

— Les messages de mise en place d'un probe point.

Les messages de configuration permettent à l'utilisateur de spécifier les informations qu'il désire collecter. Ce type de messages affecte juste une série de booléens et de structures de contrôle contenant les adresses mémoire à observer. Comme pour KDI (Kernel Debug Interface), il est possible d'observer des tâches, des 'événements, des variables et des adresses mémoire.

Pour les messages d'établissement d'un probe point, l'utilisateur envoie une adresse à surveiller.

La tâche de débogage remplacera l'instruction située à cette adresse par une instruction LCALL vers une fonction chargée de collecter les informations nécessaires (dépend des paramètres de configuration) et de les stocker dans un buffer.

En plus de prendre un instantané de l'état du noyau temps réel, cette fonction efface également le probe point en restaurant l'instruction initialement située à cette adresse.

Une fois ces fonctions exécutées, une seconde tâche chargée d'envoyer ces informations vers le PC, va passer en état actif (candidate potentielle à l'exécution). La trame de ces messages de données est bien évidemment prédéfinie et les informations qu'elle contient peuvent alors être traitées et affichées par le PC.

Après cela, le PC est chargé de renvoyer un message à la carte cible lui demandant de rafraîchir le probe point, c'est à dire de replacer une instruction LCALL à l'adresse spécifiée précédemment par l'utilisateur.

Avantages

- Placement dynamique, sans modification du code source.
- Ne consomment du temps machine que lorsque qu'elles sont activées.

Inconvénients

- Nécessitent la création et l'utilisation de deux tâches.
- Nécessitent une grande quantité de mémoire (environ 500 octets par probe point).
- L'utilisateur possède un grand nombre de paramètres pour personnaliser le débogage.

### II.1.5.2. ReTIS

**Introduction**

L'outil de débogage ReTlS a été conçu par Mikaél BRIDAY durant ses travaux pour la thèse de Doctorat de l'Université de Nantes, il permet de déboguer la simulation d'une architecture opérationnelle (le processeur et le contrôleur réseau), deux mécanismes (Trace des variables, Analyse des exécutions d'une tâche et de la pile associée) sont également implémentés. La figure suivante montre comment s'intègre l'outil par rapport à son environnement. Le débogueur est conçu pour le processeur Infineon C167 et le bus CAN.

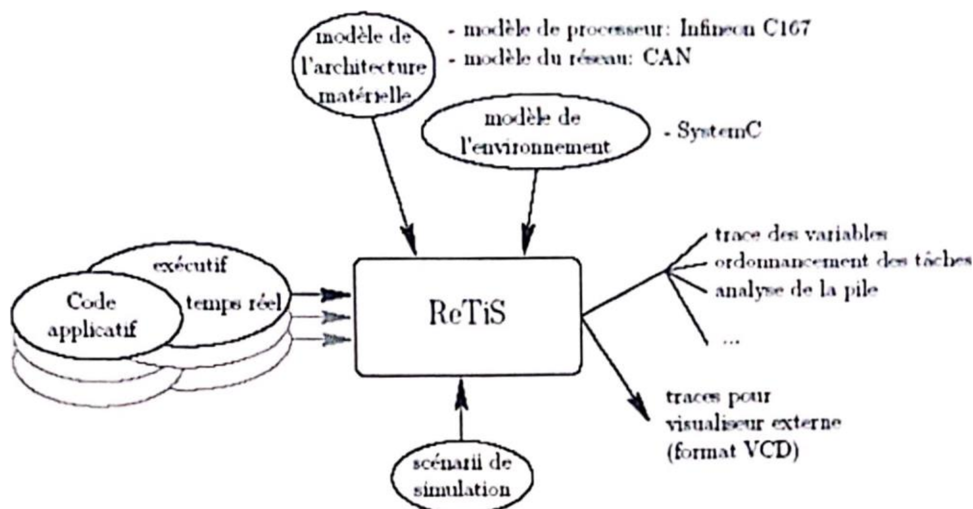


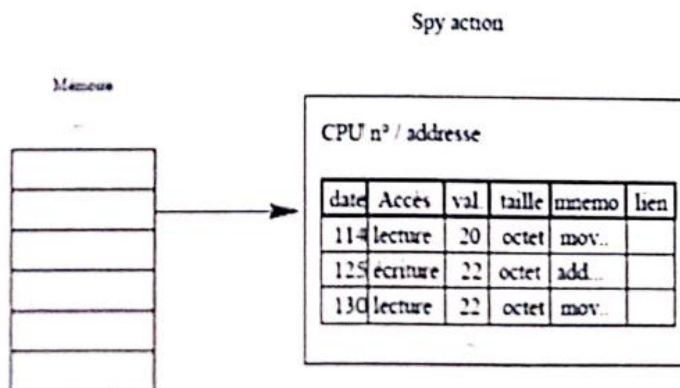

Figure II-2 : Environnement de ReTiS.

## Principe général de la trace des variables

La trace des variables est le suivi des variables d'un programme, ce suivi portant sur l'évolution des valeurs prises par ces variables, et sur les instants auxquels les variables sont utilisées, dans le but d'extraire des informations importantes pour le débogage par simulation d'un programme temps-réel.

La méthode utilisée pour "traced' les variables repose essentiellement sur les actions datées. Ces actions datées sont utilisées pour détecter les accés en lecture / écriture å un emplacement précis de la mémoire et pour exécuter en réaction une portion de code du débogueur. Afin de surveiller l'évolution des variables, ils introduisent des Spy-Action.

Chaque adresse mémoire qui est tracée est associée à un Spy- Action, celui qui enregistre tous tes accès en lecture / écriture à cette adresse dans une liste.




Un Spy-Action est une action datée spécialisée pour la trace des variables. Associé à un emplacement mémoire (processeur et adresse mémoire), un Spy-Action est activé suite à un événement sur un accès en lecture ou écriture à une mémoire et possède une liste pour enregistrer toutes les informations relatives aux accès mémoire observés.

**Avantages**

- Un mécanisme permettant d'enregistrer et de visualiser le flot de données généré par une variable. L'étude de ce flot de données peut alors permettre au développeur de systèmes temps réel d'analyser plus efficacement les causes d'un dysfonctionnement de l'application temps réel (production trop tardive de la donnée, consommation tardive par le processeur cible, etc.…)

**Inconvénients**

- Consomme toujours beaucoup de cycles machines.
- Extraction des informations très bas niveau d'un programme temps-réel.

### II.1.5.3. Trace utilité

Richard Barry (développeur du noyau FreeRTOS) a conçu un outil de traçage pendant une durée limitée et petite de l'application ; les informations collectées sont sauvegardées sur la mémoire du système embarqué pour être visualisées et analysées hors ligne. L'inconvénient majeur est que la durée de suivi de l'application est directement liée à la taille de la mémoire du système embarqué. Cet outil enregistre les séquences dans lesquelles les tâches ont été commutées. Des utilitaires sous DOS/Windows permettent de convertir le tampon rempli, de l'ouvrir et de l'examiner dans un tableur ou autre utilitaire.

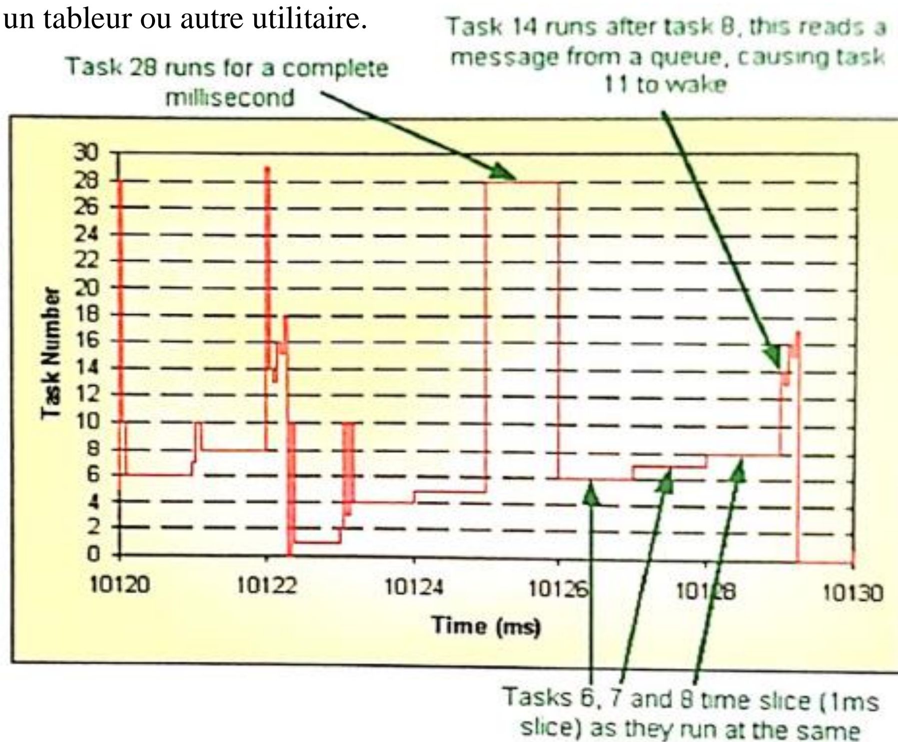



Figure II-3 : Trace sur la d'emo 186 AMD.

La figure montre la collection de données pendant 10 millisecondes.

## II.1.6. Conclusion

Le débogage d'une application temps réel n'est pas une activité simple. Une multitude d'approches existent pour contribuer au débogage du logiciel de l'application. Néanmoins, ces approches sont quelquefois spécialisées et ne contribuent pas réellement au débogage en temps réel, telles que les techniques de débogage classiques (au niveau du IDE), qui permettent d'obtenir des informations sur les registres du processeur, les variables mémoires, l'état de la pile, etc... D'autres techniques reposent sur la simulation de l'exécution du logiciel de l'application et la simulation des ressources de la machine cible.

Aussi, des outils d'analyse d'ordonnançabilité (Cheddar) permettent de vérifier l'ordonnançabilité d'un ensemble de tâches à condition que tous les paramètres de spécification soient disponibles tels que la capacité des tâches, l'échéance, la priorité, etc...

Les seules techniques, qui permettent le débogage en temps réel d'un système embarqué est celles qui emploient la connexion JTAG ou la technique ICD (ln Circuit Debugger), mais le débogage n'est pas de type RTOS-aware. En effet, le débogage consiste à obtenir seulement des informations sur les registres du processeur, les variables mémoires, l'état de la pile, etc....

# *III* ***Chapitre 3 : FreeRTOS Analyse Fonctionnelle***

# III.1. Aperçu sur FreeRTOS

## III.1.1. Caractéristiques Générales

Un système d'exploitation temps réel embarqué (Embedded RTOS) gratuit a été mis à disposition par Richard BARRY (www.FreeRTOS.org). Ce RTOS est capable de gérer un nombre infini de tâches simultanément.

Ce RTOS se prétend être portable, open source, mini noyau temps réel qui peut opérer en mode préemptif ou coopératif. Certaines des principales caractéristiques de FreeRTOS sont énumérées ci-dessous :

- **Temps réel :** En effet, FreeRTOS pourrait être un système d'exploitation temp-réel dur (hard real-time). L'assignement de l'étiquette 'Chard real time" dépendra de l'application dans laquelle FreeRTOS fonctionnera, et de la validation forte de cette étiquette dans ce contexte.
- **Préemptif ou coopératif :** L'ordonnancement peut être préemptif ou coopératif (le mode est défini par un Switch de configuration). L'ordonnancement coopératif n'implémente pas un point de décision basé sur un timer, les tâches se passent le contrôle entre elles par libération volontaire du processeur (yielding). Dans ce cas l'ordonnanceur interrompt, à une fréquence régulière, simplement pour incrémenter le tick count de l'horloge.
- **Ordonnancement dynamique :** Les points de prises de décision de réordonnancement ont lieu à une fréquence régulière de l'horloge. Les événements asynchrones (autre que le scheduler) aussi invoquent les points de décision du scheduler.
- **Algorithme d'ordonnancement :** L'algorithme d'ordonnancement est basé sur la priorité, c'est-à-dire la priorité la plus élevée d'abord ou HPF (Highest Priority First). Quand plus d'une tâche existent avec la plus grande priorité, les tâches sont exécutées en Round Robin ou à tour de rôle.
- **Communication entre les tâches :** Les tâches au sein de FreeRTOS peuvent communiquer entre elles à travers des mécanismes de synchronisation et en files d'attente.
- **Files d'attente :** La communication inter-tâches est implémentée via la création de files d'attente. La majorité de l'information échangée est passée par valeur et non par référence ce qui peut être un point à considérer pour les appels où il y a des contraintes de mémoire. Les lectures/écritures des files d'attente à l'intérieur des routines d'interruptions (ISR) sont non-bloquantes. Les lectures /écritures des files d'attente avec zéro comme valeur pour les timeouts (Timeouts nuls) sont non bloquantes, Toutes tes autres lectures/écritures sont bloquantes avec des timeouts configurables.
- **Synchronisation :** FreeRTOS permet la création et l'utilisation des sémaphores binaires. Les sémaphores eux-mêmes sont des instances spécialisées de files

d'attente de message avec une taille de 'i' et la taille de la donnée zéro. A cause de cela, la prise et ta libération de sémaphores sont opérations atomiques puisque les interruptions sont masquées et l'ordonnanceur est suspendu afin d'obtenir un verrou sur la file.

- **Blocage et prévention d'étreintes fatales (Deadlock) :** Dans FreeRTOS, les tâches sont soit des tâches non-bloquantes ou se bloquent avec un timeout c'est-à-dire pour une période de temps fixe ou bornée ou maximale. Les tâches qui se réveillent, c'est-à-dire qui sont signalées à la fin du timeout et n'ont pas eu accès à la ressource, doivent prendre ceci en considération du fait que l'API (l'appel) pour accéder à la ressource pourront retourner une notification d'échec à cette ressource. Un timeout sur chaque blocage réduit la probabilité d'apparition du deadlock.
- **Traitement des Sections Critiques :** Le traitement des sections critiques est fait en masquant les interruptions. Les sections critiques au sein d'une tâche peuvent être imbriquées. Chaque tâche maintient son propre compte d'imbrications. Cependant il est possible de forcer une commutation de contexte à l'intérieur d'une section critique (pour supporter l'ordonnancement coopératif) car les interruptions software (SWI) sont non-maskable et le forçage de réordonnancement (yield) utilise SWI pour le changement de contexte.
- **Suspension du Scheduler :** Quand l'accès exclusif est demandé au service à la MCU sans pénaliser les services de routines d'interruptions, le scheduler peut être suspendu. Suspendre le scheduler garantit que la tâche courante active ne sera pas préemptée par un événement temps réel ou multitâche tout en continuant, en même temps à servir les interruptions.
- **Allocation de la mémoire :** FreeRTOS prévoit trois (03) modèles de tas mémoire (heap memory). Le plus simple permet l'allocation de mémoire à la création de chaque tâche, mais ne permet pas de libérer cette mémoire pour la réutiliser (donc les tâches ne peuvent pas être supprimées 'deleted'). Un modèle plus complexe permet une allocation et la libération de mémoire et utilisera l'algorithme (best-fit) pour allouer de la mémoire à partir du tas (heap). Cependant l'algorithme ne permet pas la combinaison de segments adjacents libres de mémoire. Le plus complexe de ces modèles fournit un motif de conception (wrappers) pour malloc. Un algorithme propre à l'utilisateur peut être créé pour répondre aux exigences de chaque application.
- **Inversion de priorité :** FreeRTOS implémente des outils pour l'héritage de priorité afin de résoudre le problème d'inversion de priorité.

### III.1.2. Distribution du code source

FreeRTOS est distribué en tant que code ayant une arborescence comme montrée dans la figure III-1. FreeRTOS inclut le code source indépendant de la plateforme cible dans le répertoire Source. La majorité des fonctionnalités de FreeRTOS est prévue dans les fichiers **tasks.c, queue.c, list.c**, et **coroutines.c** (et les fichiers d'entête associés).

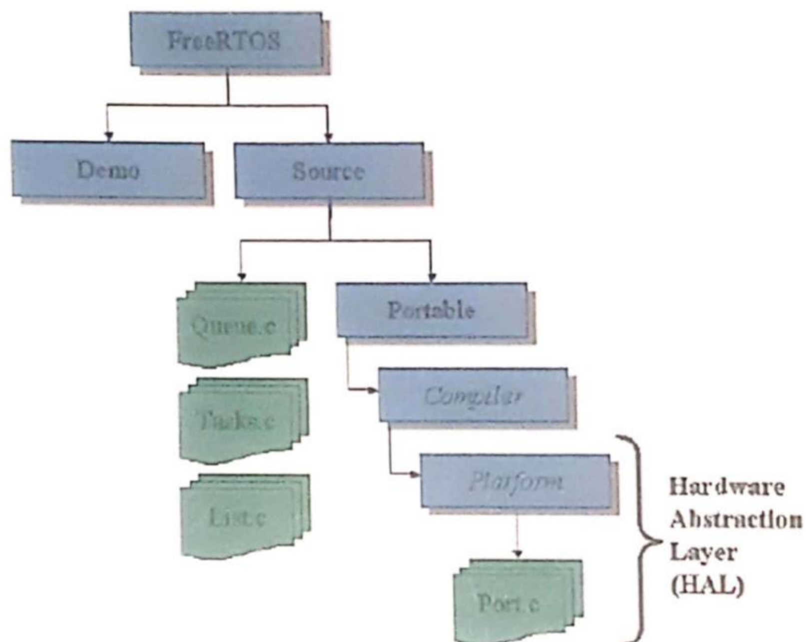



Figure III-1 : FreeRTOS Source Distribution

FreeRTOS fournit une couche d'abstraction du hardware (Hardware Abstract Layer — HAL) ayant pour cible une variété de combinaisons « compilateurs/hardware ». Les fonctionnalités spécifiques à une architecture pour chaque compilateur/hardware cible est prévue dans les fichiers port.c et portmacro.h au sein de la couche HAL. Toutes les fonctions référencées dans ce mémoire qui ont le préfixe port » appartiennent à la couche HAL et sont implémentées dans l'un des fichiers portables.

Le dossier Demo fournit des exemples de code pour plusieurs applications de démonstration. Ce dossier est organisé de la même façon que le dossier Portable car les démonstrations sont écrites et compilées pour fonctionner sur certaines architectures utilisant des compilateurs divers.

# III.2. Description Du Noyau FreeRTOS

## III.2.1. Introduction

Cette séance est une description détaillée du noyau FreeRTOS.

L'analyse et la description débutera par une revue sur les items de configuration des pre-éxecution. Puis suivi du Gestionnaire de Tâches et Gestionnaire de listes, on détaillera après sur le scheduler et les mechamsmes des files d'attente par le gestionnaire des files d'attentes.

## III.2.2. Configuration de FreeRTOS

Les opérations de FreeRTOS sont configurées d'une manière significative dans le fichier **FreeRTOSConfig.h**. Ce fichier est décrit dans la liste ci-dessous

```
        FreeRTOS.org V5.0.0 - Copyright (C) 2003-2008 Richard Barry.
        This file is part of the FreeRTOS.org distribution.

*/

        FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H
#include <p18cxxx.h>
/*-----------------------------------------------------------
 *      Application specific definitions.
 *      These definitions should be adjusted for your particular hardware and
 * appl i cation requirements.
 *THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS org WEB SITE.
    ----------------------------------------------------------*/

#define configUSE-PREEMPTION              1
#define configUSE-1DLE-HOOK               0
#define configlJSE-TICK-HOOK              0
#define configTICK-RATE-HZ                        ((portTickType ) 1000 )
#define configCPU-CLOCK_HZ                        ((unsi gned portLONG ) 16000000 )
#define configMAX_PRIORITIES                  ( (unsigned portBASE—7PE ) 10 )
#define confiGMINIMAL-STACK-SIZE            (105 )
#define configTOTAL-HEAP_SIZE                 (( size-t ) 1024 )
#define configMAX_TASK_NAME_LEN           (4)
#define configlJSE-TRACE_FACILITY          0
#define configUSE_16_BIT_TICKS            1
#define configlDLE-SHOULD-YIELD           1

/* Co-routine defi nitions.*/        0
#define configUSE_CO_ROUTINES
#define configMAX—CO-ROUTINE-PRIORITIES ( 2 )
/e Set the following definitions to 1 to include the API function, or zero to exclude the API function.

#define INCLUDE—vTaskPrioritySet          0
#define INCLUDE_uxTaskPriorityGet         0
#define INCLUDE—vTaskDe1 ete                     1
#define INCLUDE_vTaskC1 eanUpResources    0
#define INCLUDE_vTaskSuspend              0
#define INCLUDE_vTaskDe1 ayUnti1          1
#define INCLUDE—vTaskDe1ay                       1

#endif /* FREERTOS-CONFIG-H*/
```



Figure III-2: FreeRTOSConfig.h (PIC18)

- **configUSE_PREEMPTION:** Mis à 1 permet de sélectionner l'ordonnancement préemptif. Mis à 0 permet de sectionner l'ordonnancement coopératif.
- **configUSE_IDLE_HOOK:** Un crochet (hook) nommé "Idle Task Hook" s'exécute pendant chaque cycle du processeur Idle (inactif). Cette variable est mise à « 1 » si les hooks sont utilisés ou désirés. La fonction s'exécutera à la priorité du processus idle. Pour des besoins d'instrumentation « probing » un hook de type idle devient idéal
- **configUSE_TICK_HOOK:** Une fonction de type TICK HOOK s'exécutera à chaque interruption du tick du noyau si cette variable est mise à 1. Très utile pour des besoins d'instrumentation.
- **configCPU_CLOCK_HZ:** C'est l'horloge interne du processeur
- configTICK_RATE HZ: C'est la fréquence sur laquelle le tick de FreeRtos fonctionnera. Elle est mise à 1000 Hz dans la plupart des cas (la plupart des Demos)
- **configMAX_PRIORITIES:** Le nombre total de niveaux de priorités qu'on peut avoir quand on hiérarchise les tâches. Chaque nouveau niveau de priorité induit une nouvelle liste créée par le noyau, donc les cibles à forte contrainte de mémoires doivent minimiser le nombre de niveaux de priorité
- **configMAX_ TASK_NAME_LEN:** La taille maximale permise (# caractères) pour le nom descriptif donné à une tâche quand celle-ci est créée. Ce nom est généralement utile pour le débogage et la visualisation du système.
- **configUSE_16 BIT_TICKS:** Le temps est mesure en 'ticks', c'est le nombre de ticks d'interruptions qui a été exécuté depuis le démarrage du noyau. Mis à 1 cause portTickType à être définit en 16 bits non-signés. Mis à 0, portTickType est définit à 32 bits non-signés.
- **configTICK_RATE_HZ:** La fréquence du tick d'interruption de FreeRTOS. Cependant une plus grande fréquence du tick implique que le temps est mesuré avec une plus grande résolution. Le noyau utilisera donc, beaucoup de temps CPU et diminuera par conséquent, son efficacité.
- **configIDLE_SHOULD_YIELD:** Ce paramètre contrôle le comportement des tâches à la priorité Idle (basse). Il n'a d'effet que si :

- L'ordonnancement préemptif est utilisé.
- L'utilisateur créé des tâches avec une priorité idle

Si les tâches partagent la même priorité, et aucune d'elles n'a été préemptée, alors l'ordonnanceur alloue des quantités de temps à parts égales entre elles.
Si le paramètre est défini à 0, la tâche avec la priorité Idle n'est pas préemptée avant la fin de son quota de temps qui lui est réservée. Si le paramètre est défini à 1, la tâche de priorité Idle donnera la main à la tâche de plus forte priorité, toutefois cette tâche ne prendra de temps que ce qui rester de la tâche liberée (préemptée), c'est-à-dire elle n'aura pas toute la tranche de temps normalement allouée.

- **configUSE_CO_ROUTINES :** Mis à 0 si on n'utilise pas de co-routines
- **configMAX_CO_ROUTINE_PRIORITIES:** Le nombre de priorités permit aux coroutines d'une application. Les tâches sont 'prioritisées' séparément
- **configUSE_TRACE_FACILITY:** Mis à 1 pour permettre les fonctionnalités de traçages et visualisation, dans ce cas il faut prévoir un buffer de trace.
- **configMINIMAL_STACK_SIZE:** La taille de la pile utilisée par la tâche Idle. En général, cela ne devrait pas être réduit de la valeur définie dans le fichier FreeRTOSConfig.h fournie avec la Demo pour le port utilisé.
- **configTOTAL_HEAP_SIZE:** Cette configuration détermine la taille mémoire RAM qui sera utilisée par FreeRTOS pour les piles, les blocs de contrôle des tâches, les listes et les files d'attente. Ceci est la taille mémoire disponible à l'allocation à partir de tas selon les méthodes d'allocation.

### III.2.3. Gestionnaire de Tâches

Cette section décrit les structures du gestionnaire de tâches et les mécanismes utilisées par le scheduler

#### III.2.3.1. Bloc de contrôle d'une tâche

Le noyau FreeRTOS gère les tâches via leur bloc de contrôle (Task Control Block TCB). Un TCB existe pour chaque tâche dans le noyau et contient toutes les informations nécessaires pour décrire complètement l'état d'une tâche. Les champs dans le TCB pour FreeRTOS sont montrés ci-dessous (dérivant de task.c)

| | |
|---|---|
| Top of Stack | Pointeur sur le dernier item placé dans la pile pour cette tache |
| Task State | Un item de type liste qui place le TCB dans la file des prêtes ou bloquées |
| Event List | Un item de type liste utilisé pour placer le TCB dans la file d'attente d'événements (Event List) |
| Priority | Priorité de la tâche (0= la plus basse) |
| Stack Start | Pointeur vers le début de la pile de la tâche |
| TCB Number | Un champ pour le débogage et le traçage |
| Task Name | Nom de la tâche |
| Stack Depth | La taille totale de la pile en variables (non en octet) |

Figure III-3 : TCB de FreeRTOS

### III.2.3.2. Diagramme d'état de la tâche

Dans FreeRTOS, une tâche peut exister en cinq (05) états ; Supprimée (Deleted),Suspendue (Suspended), Prête( Ready),Bloquée ( Blocked) et En exécution( Running) comme décrit dans

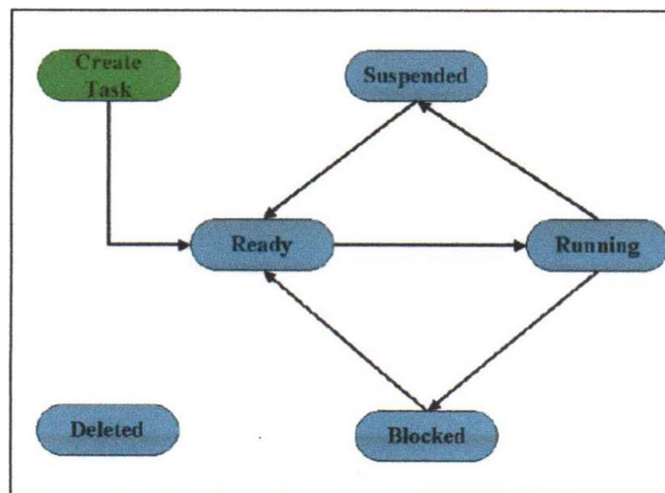



Figure III-4 : Diagramme d'état d'une tâche dans FreeRTOS

Le noyau FreeRTOS crée une tâche en instantiant et déclarant unTCB. Les nouvelles tâches sont immédiatement dans l'état des Prêt en les ajoutant à la liste des prêtes (Ready list). La liste des « prêtes » est ordonnée selon la priorité avec le fait que les tâches de même priorité sont exécutées en RRS. L'implémentation de FreeRTOS utilise une file d'attente multiple des tâches prêtes -Une file par niveau de priorité. Quand il s'agit de sélectionner la première tâche à exécuter, le scheduler commence avec le niveau le plus prioritaire et poursuit son chemin en

descendant vers le niveau bas. FreeRTOS ne possède pas de files de tâches en exécution. Puisque en fait, le noyau maintient la variable pxcurrentTCB pour
identifier la tâche dans la liste des prêtes, qui est en exécution. PxcurrentTCB est donc défini comme un pointeur vers la structure TCB).
Les tâches dans FreeRTOS peuvent se bloquer lors de l'accès à une ressource si celle-ci n'est pas disponible, Le scheduler bloque les tâches seulement quand elles tentent de lire ou d'écrire dans la file d'attente qui est soit vide soit pleine respectivement. Ceci est valable pour les sémaphores lors de leur acquisition puisqu'ils sont un cas spécial de files d'attente. Comme indiqué, précédemment, l'accès aux files d'attente peut être bloquant ou non bloquant. La distinction est faite via la variable xTicksToWait qui est passée comme argument dans l'appel d'accès à la file. Si xTicksToWait est 0, et la file est vide /plelne la tâche ne se bloque pas. Autrement, la tâche se bloquera pour une période de xTicksToWait ticks ou jusqu'à un événement sur la file débloquera la ressource. Les tâches peuvent se bloquer volontairement pour des périodes de temps via l'API. Le scheduler utilise la lise des tâches retardées « delayed » dans ce cas. Le scheduler visite cette lise des « delayed » à chaque point de décision d'ordonnancement pour déterminer si l'une des tâches a un timeout. Les tâches ayant des timeouts sont mises à la liste des prêtes. L'API de FreeRTOS fournit les fonctions vTaskDelay et vTaskDelayUntil qui peuvent être utilisées pour mettre un tâche dans la liste « delayed list ».
Chaque tâche, ou en fait toutes les tâches à l'exception de celle en exécution (et celles servant les ISRs) peuvent être mises à l'état des suspendues « Suspended State » indéfiniment. Les tâches mises dans cet état ne sont pas en attente d'événements et ne consomment aucune ressource ou une attention du noyau. Quand les tâches mises à l'état non-suspendues, elles retournent à l'état des prêtes.
Les tâches terminent leur cycle de vie en les supprimant (ou se suppriment elles-mêmes).
L'état des supprimées « Deleted state » est nécessaire puisque la suppression des tâches n'implique pas la libération des ressources immédiate utilisées par ces tâches. En mettant les tâches en « Deleted state », le scheduler dans le noyau FreeRTOS est amené à ignorer ces tâches. La tâche IDLE a la responsabilité du nettoyage de la mémoire après la suppression des tâches, cette opération risque de prendre du temps car IDLE à la priorité la plus basse.

### III.2.4. Gestion des listes

Cette section nous donne un aperçu sur la création et la gestion des listes dans FreeRTOS. Ces informations sont utiles pour comprendre les fonctionnalités des différents modules dans FreeRTOS.

### III.2.4.1. Listes « Ready » et « Blocked »

La figure III-5 montre toutes les listes qui sont créées et utilisées par le scheduler et leur dépendance avec Iles valeurs de configuration dans FreeRTOSConfig.h

| **FreeRTOSConfig.h** | **List Created** |
|---|---|
| **configMAX_PRIORITIES** | ReadyTasksLists[0]<br><br>Ready TasksList[configMAX_PRIORITIES] |
| **INCLUDE_vTaskDelete == 1** | TasksWaitingTermination |
| **INCLUDE_vTaskSuspend ==1** | SuspendedTaskList |
| **N/A** | PendingReadyList |
| **N/A** | DelayedTaskList |
| **N/A** | OverflowDelayedTaskList |

Figure III-5: Listes Créées par le Scheduler

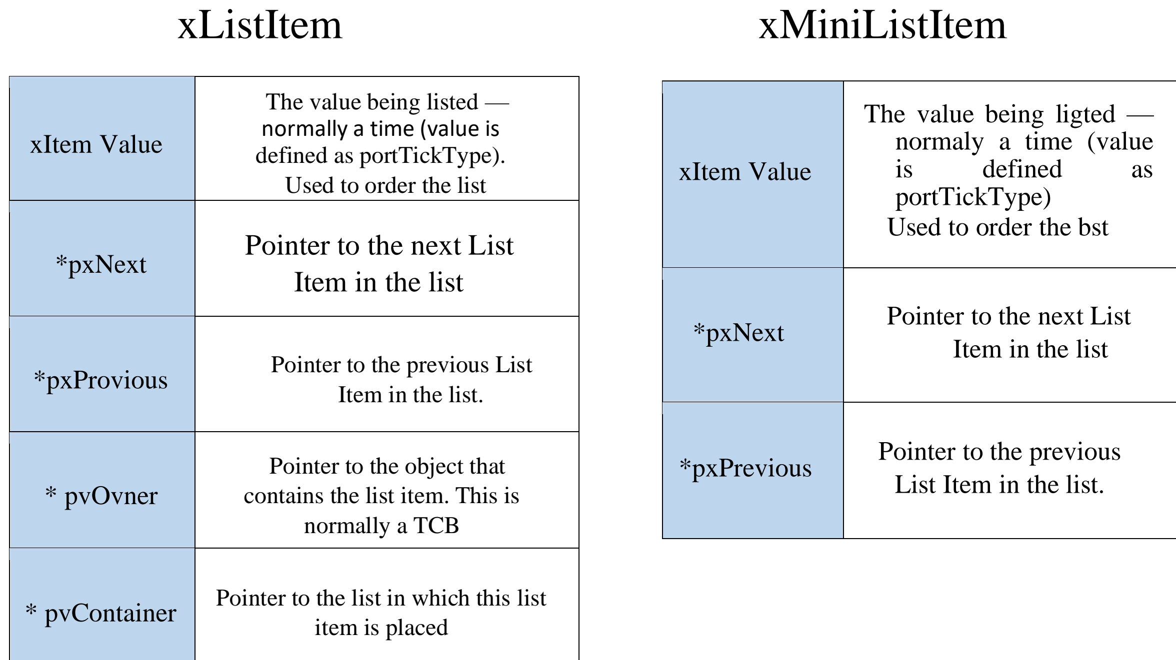
Noter que la liste des {Ready} n'est pas une liste singuliere mais n listes où
n=configMAX_PRIORITIES
Chacune des listes dans la Figure III-5 est créé comme type xList. Qui est une structure définie comme détaillé dans la Figure III-6.

| NumberOfItems | The number of items in the list. |
|---|---|
| (xListItem) *pxIndex | Pointer used to walk through the list. It points to successive list items in the list |
| (xMiniListItem) xListEnd | A list item that marks the end ofthe list. It contains the maximum value in xItemValue and therefore always appears at the end of the list. |

Figure III-6: Type xList

Chaque liste a une entrée identifiant le nombre d'items dans la liste. La liste possède un index de type pointeur pxlndex qui pointe vers un des items dans la liste (et qui est utilisé pour iterer à travers la liste). Le pointeur pxlndex pointe sur le type xListltem qui est le seul que la liste peut contenir. La seule exception est xListEnd qui est du type xMiniListltem.

Les structures xListltem et xMiniListltem sont détaillées dans la Figure III-7.



xListItem

| | |
|---|---|
| xItem Value | The value being listed — normally a time (value is defined as portTickType). Used to order the list |
| *pxNext | Pointer to the next List Item in the list |
| *pxProvious | Pointer to the previous List Item in the list. |
| * pvOvner | Pointer to the object that contains the list item. This is normally a TCB |
| * pvContainer | Pointer to the list in which this list item is placed |

xMiniListItem

| | |
|---|---|
| xItem Value | The value being ligted — normaly a time (value is defined as portTickType) Used to order the bst |
| *pxNext | Pointer to the next List Item in the list |
| *pxPrevious | Pointer to the previous List Item in the list. |

Figure III-7: Types xList

### III.2.4.2. Initialisation des listes :

La Figure III-8 montre un exemple d'initialisation de la liste {DelayedTaskList}. Le nombre d'items (numberOfltems) est initialement mis à zéro. Les pointeurs pxlndex , pxNext et pxPrevious pointent tous vers l'adresse de la structure de xListEnd ( c.-à-d. pxlndex=pxNext=pxPrevious=&(xListEnd))

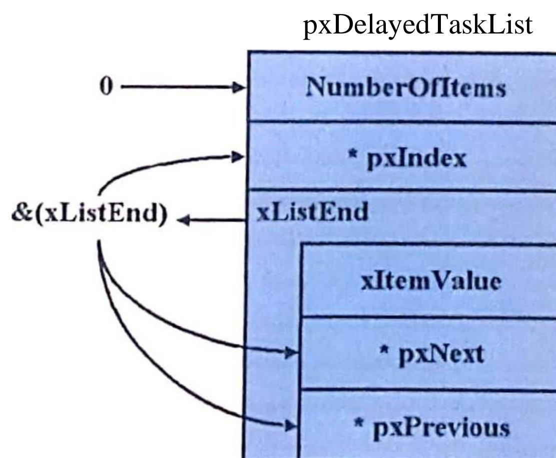



Figure III-8 : Initialisation de liste

xltemValue dans la structure xListEnd doit contenir la plus grande valeur possible. Car {DelayedTaskList} est utilisée pour chainer les tâches selon la durée du timout, cette valeur est définie comme étant égale à portMAX_DELAY.

### III.2.4.3. Insertion dans une liste.

Pour insérer une tâche dans une liste (par exemple, (DelayedTaskList)), FreeRTOS utilise la fonction vListInsert. Les arguments de cette fonction incluent le pointeur vers la liste en question et un pointeur vers le membre Genericlistlem du TCB à insérer comme illustré dans la Figure III-9.

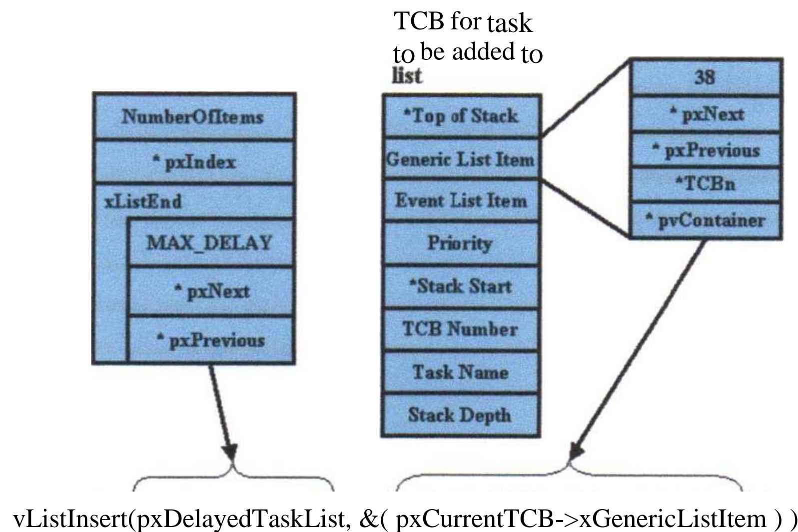



Figure III-9: ***vlistInsert*** avec Arguments

Dans la Figure III-9, xltemValue dans le champ GEnericListltem a déjà été mis à 38 (nombre arbitraire). Dans ce cas, ceci représente la valeur absolue en ticks d'horloge au bout de laquelle la tâche associée à ce TCB sera réveillée puis réinsérer dans la liste des Prêtes {Ready List}. Aussi, il faut noter que le pointeur *pvOwner a été affecté pour pointer vers le TCB contenant le GenericListltem. Ceci permet une identification rapide du TCB.

La Figure III-10 montre un exemple de ce que la liste {DealyedTaskList) pourrait ressembler avec le chainage de deux tâches. Le pointeur *pxNext dans la structure xListEnd de notre liste n'est pas NULL—il pointe vers la première entrée dans la liste comme montré dans la Figure.

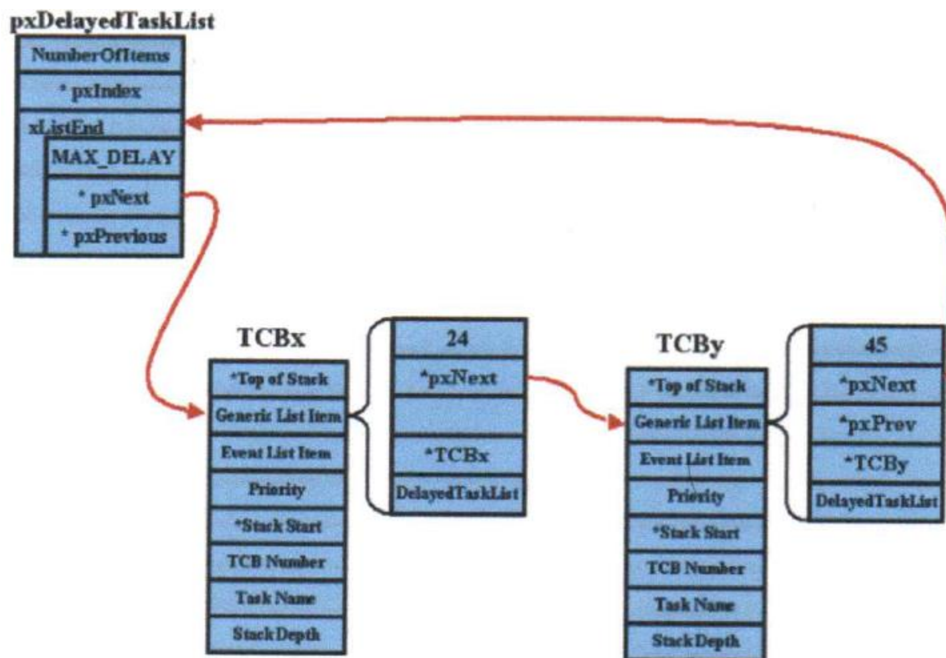



Figure III-10: DelayedTaskList

Pour insérer une nouvelle tâche dans {DalyedTaskList}, la fonction vListlnsert procède comme suit. xltemValue au sein de Generic List Item est comparée avec xltemValue du premier TCB dans la liste. Dans le cas de {DealyedTaskList}, c'est la valeur absolue en ticks d'horloge au bout de laquelle la tâche sera reveillée. Si cette valeur est inférieure (Dans ce cas 24<38), le pointeur *pxNext est utilisé pour se deplacer sur le TCB suivant dans la liste, Dans le cas contraire, Le TCB courant doit être déplacé vers la droite' quand le nouveau TCB de la tâche est inséré. Ceci est fait en modifiant les pointeurs *pxPrev et *pxNext des items adjacents de la liste et les deux pointeurs *pxPrev et *pxNext au sein du nouveau TCB Lui-même. Et en fin le pointeur *pxContainer du nouveau TCB inséré pointera vers la liste {Do/yedTaskList}. Ce pointeur est apparemment utilisé pour une éventuelle suppression rapide plus tard. Une fois le nouveau TCB inséré, la valeur NumberOfltems de la structure {DalyedToskList} est mise å jour.

Le code qui implémente cette insertion est montré dans la figure III-11

```
void vListInsert( xList *pxList, xListItem *pxNewListItem )
{
volatile xListItem *pxIterator;
portTickType xValueOfInsertion;

	/* Insert the new list item into the list, sorted in ulListItem order. */
	xValueOfInsertion = pxNewListItem->xItemValue;

	/* If the list already contains a list item with the same item value then
	the new list item should be placed after it.  This ensures that TCB's which
	are stored in ready lists (all of which have the same ulListItem value)
	get an equal share of the CPU.  However, if the xItemValue is the same as
	the back marker the iteration loop below will not end.  This means we need
	to guard against this by checking the value first and modifying the
	algorithm slightly if necessary. */
	if( xValueOfInsertion == portMAX_DELAY )
B	{		pxIterator = pxList->xListEnd.pxPrevious;
```

```
        }
        else
        {
                /* *** NOTE
*************************************************************
                If you find your application is crashing here then likely causes are:
                        1) Stack overflow -
                           see http://www.freertos.org/Stacks-and-stack-overflow-
                           checking.html
                        2) Incorrect interrupt priority assignment, especially on Cortex
                           M3
                           parts where numerically high priority values denote low
                           actual
                           interrupt priories, which can seem counter intuitive.  See
                           configMAX_SYSCALL_INTERRUPT_PRIORITY on
                           http://www.freertos.org/a00110.html
                        3) Calling an API function from within a critical section or
                          when
                           the scheduler is suspended.
                        4) Using a queue or semaphore before it has been initialised or
                           before the scheduler has been started (are interrupts firing
                           before vTaskStartScheduler() has been called?).
                           See http://www.freertos.org/FAQHelp.html for more tips.

        **********************************************************************/
A               for( pxIterator = ( xListItem * ) &( pxList->xListEnd );

                     pxIterator->pxNext->xItemValue <= xValueOfInsertion;
                   pxIterator = pxIterator->pxNext )
                {
                        /* There is nothing to do here, we are just iterating to the
                        wanted insertion position. */
                }
        }

        pxNewListItem->pxNext = pxIterator->pxNext;
        pxNewListItem->pxNext->pxPrevious = ( volatile xListItem * ) pxNewListItem;
        pxNewListItem->pxPrevious = pxIterator;
        pxIterator->pxNext = ( volatile xListItem * ) pxNewListItem;

        /* Remember which list the item is in.  This allows fast removal of the
        item later. */
        pxNewListItem->pvContainer = ( void * ) pxList;

        ( pxList->uxNumberOfItems )++;
```

Figure III-11 : Code Extrait de List.c dans FreeRTOS

La boucle for au point A initialise px/terator (ayant le type Listltem) au dernier item de la liste qui est, par défaut, ListEnd. Comme mentionné, le pointeur *pxNext de ListEnd pointe au premier item dans la liste. L'opération de comparaison dans la boucle for vérifie la donnée xltemValue dans la structure pointé par *pxNext courant si le résultat est vrai, pxlterator prend la valeur de l'Item suivant dans la liste.

Il faut noter qu'une condition limite a lieu quand le nouveau xltemValue (à insérer) est égale à portMAX_DELAY comme défini dans FreeRTOSConfig.h (FreeRTOS prend en charge cette exception au point B du code (Figure III-11) en insérant la tâche à la seconde dernière place dans la liste.

## La taille du compteur du Timer et {DelayedTaskList}

Les tâches sont placées dans la file d'attente {DelayedTaskList) par le scheduler ou par des appels API (vTaskDelay et vTaskDelayUntil). Dans tous les cas, un temps absolu est calculé au bout duquel la tâche doit être réveillée. Par exemple, pour un délai de 10 ticks, alors 10 est ajouté au tickcount courant et devient xltemValue à enregistrer dans la structure GenericListltem.

Cependant, les contrôleurs embarqués ciblées par FreeRTOS possèdent des compteurs qui peuvent être de petite taille comme 8 bits —causant une remise à zéro à chaque 255 ticks. Pour résoudre ce problème, FreeRTOS utilise deux listes de delay - {DelayedTaskList) et {OverflowDelayedTaskList}.

Comme illustré dans la figure III-12, le temps delay est additionné au temps courant en A. En B, si la valeur somme retournée est inferieur à la valeur du timer courante, alors le temps du réveil (TimeToWake) devra être inséré dans {OverflowDelayedTaskList).

```
/* Calculate the time to wake - this may overflow but this is
                        not a problem. */
                        xTimeToWake = xTickCount + xTicksToDelay;        A

                        /* We must remove ourselves from the ready list before
Adding ourselves to the blocked list as the same list item is used for both lists.
*/
                        vListRemove( ( xListItem * ) &( pxCurrentTCB-
>xGenericListItem ) );

                        /* The list item will be inserted in wake time order. */
                        listSET_LIST_ITEM_VALUE( &( pxCurrentTCB-
>xGenericListItem ), xTimeToWake );

                        if( xTimeToWake < xTickCount )        B
                        {
                              /* Wake time has overflowed.  Place this item in
                                 the overflow list. */
                              vListInsert( ( xList * ) pxOverflowDelayedTaskList,
( xListItem * ) &( pxCurrentTCB->xGenericListItem ) );
                        }
                        else
                        {
                              /* The wake time has not overflowed, so we can use
                                 the current block list. */
                              vListInsert( ( xList * ) pxDelayedTaskList, (
xListItem * ) &( pxCurrentTCB->xGenericListItem ) );
                        }
                  }
```

Figure III-12 : Quelle liste à utiliser ? (de **Tasks.c**)

Noter que, pour que ça marche, le nombre maximal de ticks utilisé pour bloquer une tâche doit être inférieur à la taille du compteur (ex : FF dans le cas d'un compteur 8 bits). Cette valeur est définie par la variable portMAX_DELAY dans **FreeRTOSConfig.h.**

A chaque incrémentation du tickcount (dans la fonction vTaskIncrementTick), une vérification de débordement du timer est opérée. Si c'est le cas, les pointeurs vers {DelayedTaskList} et {OverflowDelayedTaskList} sont inversés (Figure III-13).

```
/* Called by the portable layer each time a tick interrupt occurs.
Increments the tick then checks to see if the new tick value will cause any
tasks to be unblocked. */
if( uxSchedulerSuspended == ( unsigned portBASE_TYPE ) pdFALSE )
{
        ++xTickCount;
        if( xTickCount == ( portTickType ) 0 )
        {
                xList *pxTemp;

                /* Tick count has overflowed so we need to swap the delay lists.
                If there are any items in pxDelayedTaskList here then there is
                an error! */
                pxTemp = pxDelayedTaskList;
                pxDelayedTaskList = pxOverflowDelayedTaskList;
                pxOverflowDelayedTaskList = pxTemp;
                xNumOfOverflows++;
        }

        /* See if this tick has made a timeout expire. */
        prvCheckDelayedTasks();
}
```

Pointeurs de listes inversés

Figure III-13 : Inverser les pointeurs de listes lors d'un débordement du timer

### III.2.5. L'Ordonnanceur de FreeROS (FreeRTOS Scheduler)

Cette section fournit un aperçu détaillé du mécanisme d'ordonnancement dans FreeRTOS. Vu que les options de configurations permettent les modes préemptif et coopératif, le mécanisme d'ordonnancement est considérablement complexe.

La Figure III-14 nous donne un aperçu sur l'algorithme du scheduler. Le scheduler agit comme un ISR (timer Interrupt Service Routine) **-vPortTickInterrupt** qui est activé une fois à chaque période tick. La période tick est défini en configurant le paramètre configTICK_RATE HZ dans **FreeRTOSConfig.h.**

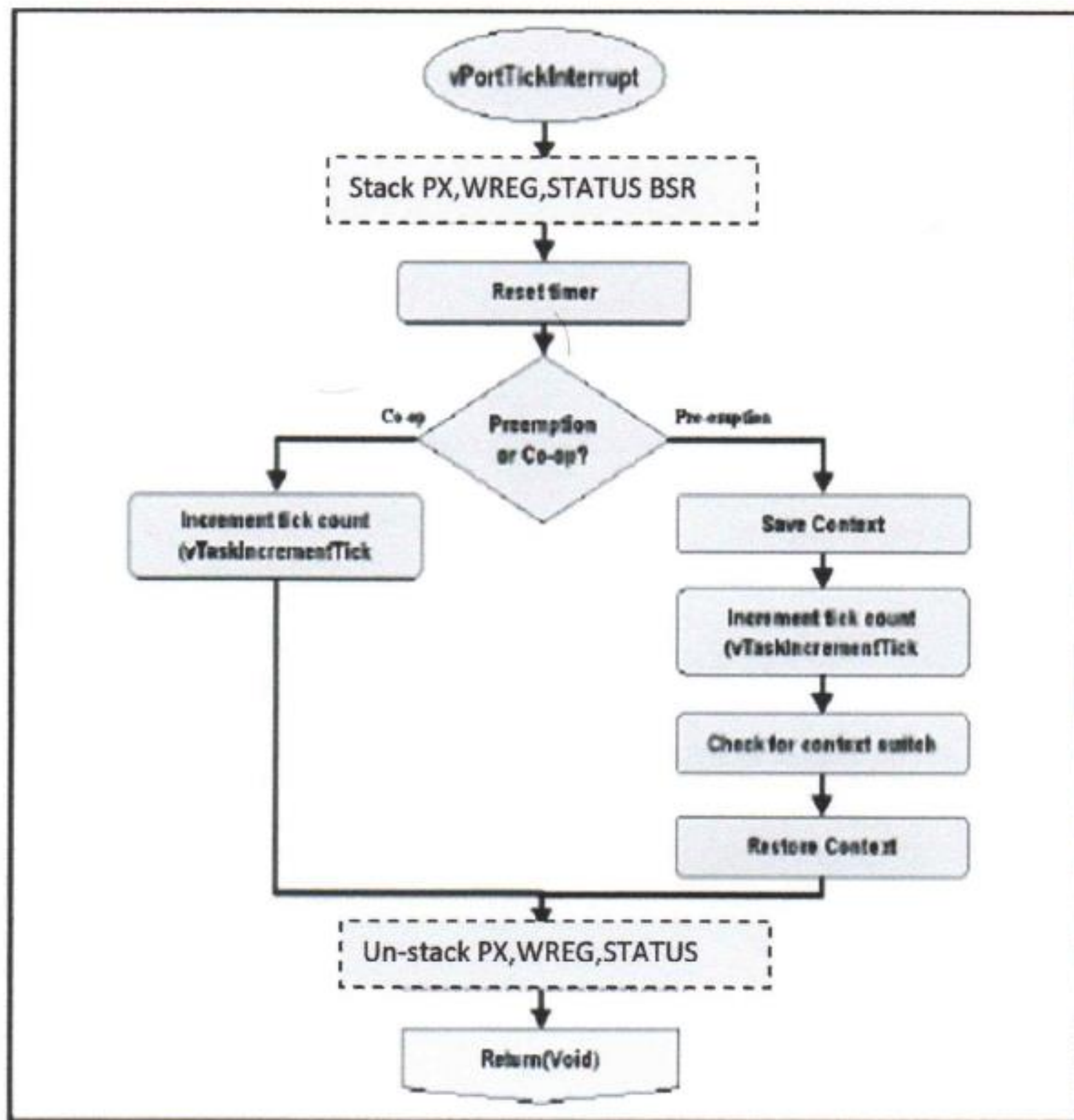



Figure III-14 : Scheduler Algorithm

Puisque le Scheduler opère comme une interruption, c'est donc une partie du HAL et contient un code spécifique d'implémentation. Dans la Figure III-14, l'implémentation du HAL pour le PIC18XXX inclut empilement (et dépilement) d'un nombre de registres software (en pointillés Fig III-14).

La première opération exécutée par le Scheduler est de réinitialiser le compteur du timer

Une instruction hardware spécifique) dans le but de démarrer la prochaine période du tick. FreeRTOS peut être configuré soit en mode coopératif ou préemptif. Dans le Scheduler, après la remise à zéro de l'horloge, la variable config_USE PREEMPTION contenue dans FreeRTOSConfig.h détermine le mode d'ordonnancement utilisé.

Dans le mode coopératif, la seule opération faite avant le retour de la routine d'interruption du timer est d'incrémenter le tick count. Il y a certaines considérations logiques à prendre en considération derrière cette opération pour traiter des cas spéciaux et les limitations de la taille du timer.

Si le mode est préemptif, alors la première opération et d'empiler le contexte de la tâche courante dans le cas où une commutation de contexte est requise. Le Scheduler incrémente le tick count et ensuite vérifie si cette action a causé le déblocage d'une tâche. Si une tâche était débloquée et que sa priorité est supérieure à celle de la tâche courante, alors une commutation de contexte est exécutée. En dernière étape, le contexte est restauré, et le Scheduler fait un retour d'interruption.

### III.2.5.1. Le cadre du contexte de la tâche

Les paragraphes suivants décrivent la construction du cadre du contexte de FreeRTOS et le mécanisme par lequel un changement de contexte est exécuté. Le contexte de la tâche est construit à partir de données fournies automatiquement en tant qu'un service d'interruption ainsi que d'autres informations additionnelles de plusieurs macros, Il est important de savoir ce qui est attendu au sein du cadre de contexte et comment remplir celui-ci quand en démarre une tâche et quand on fait une commutation de tâches.

Quand une routine d'interruption se produit, elle empile immédiatement

(Automatiquement) le contexte d'exécution en utilisant le pointeur de pile (Stack Pointer SP). Dans la plupart des microcontrôleurs et des processeurs, le contexte consiste du compteur de programme (PC adresse de retour) et du registre d'état et d'autres registres de la machine. Tous ces registres sont empilés dans l'ordre avant de sauter à la routine de service d'interruption. La figure III-15 montre une tâche, Tâche avec son TCB correspondant et la pile avant que l'interruption se produise et immédiatement après l'interruption et avant que l'ISR prenne contrôle du processeur.

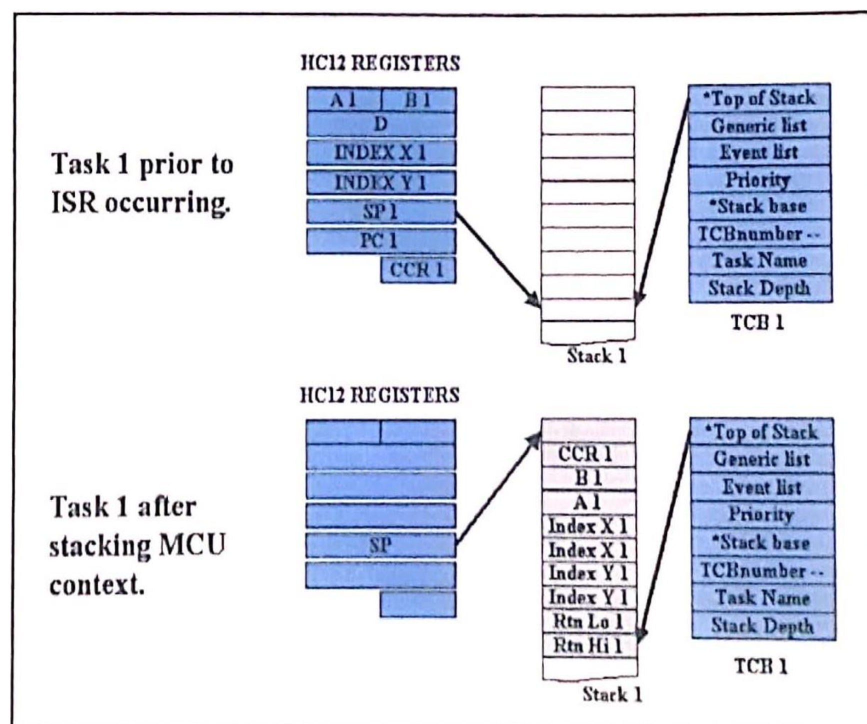



Figure III-15: Stacking of MCU context

Le reste des registres sera sauvegardé (empilé) par une routine de service d'interruption *portSAVE_CONTEXT()*

Pour quitter, la routine d'interruption continue par la restauration du contexte de la tâche suivante et par l'exécution de l'instruction de retour d'interruption RTI.

### III.2.5.2. Démarrage et arrêt des tâches

Bien que démarrer ou arrêter une tâche n'est pas une fonction directe du Scheduler, une brève description sera donnée ci-dessous, car le Scheduler manipule les structures des données et les piles créées quand une tâche est créée, par conséquent, la compréhension de la création et la destruction des tâches aidera dans la description des fonctions restantes du Scheduler.

Les tâches sont créées en invoquant *xTaskCreate()* au sein de **main.c** (création statique des tâches) ou dans le corps (code) d'une tâche (création dynamique des tâches).

Les paramètres requis pour la création d'une tâche incluent :

- Un pointeur vers la fonction qui implémente la tâche (corps). Pour des raisons évidentes, le code qui implémente la fonction tâche doit être une boucle infinie.
- Un nom pour la tâche. Ceci est utilisé principalement pour débogage et monitoring dans FreeRTOS.
- La profondeur de la pile de la tâche.
- La priorité de la tâche,
- Un pointeur vers tout paramètre dont a besoin la fonction de la tâche.

Une vue d'ensemble du processus de création d'une tâche est montrée dans la Figure III-16.

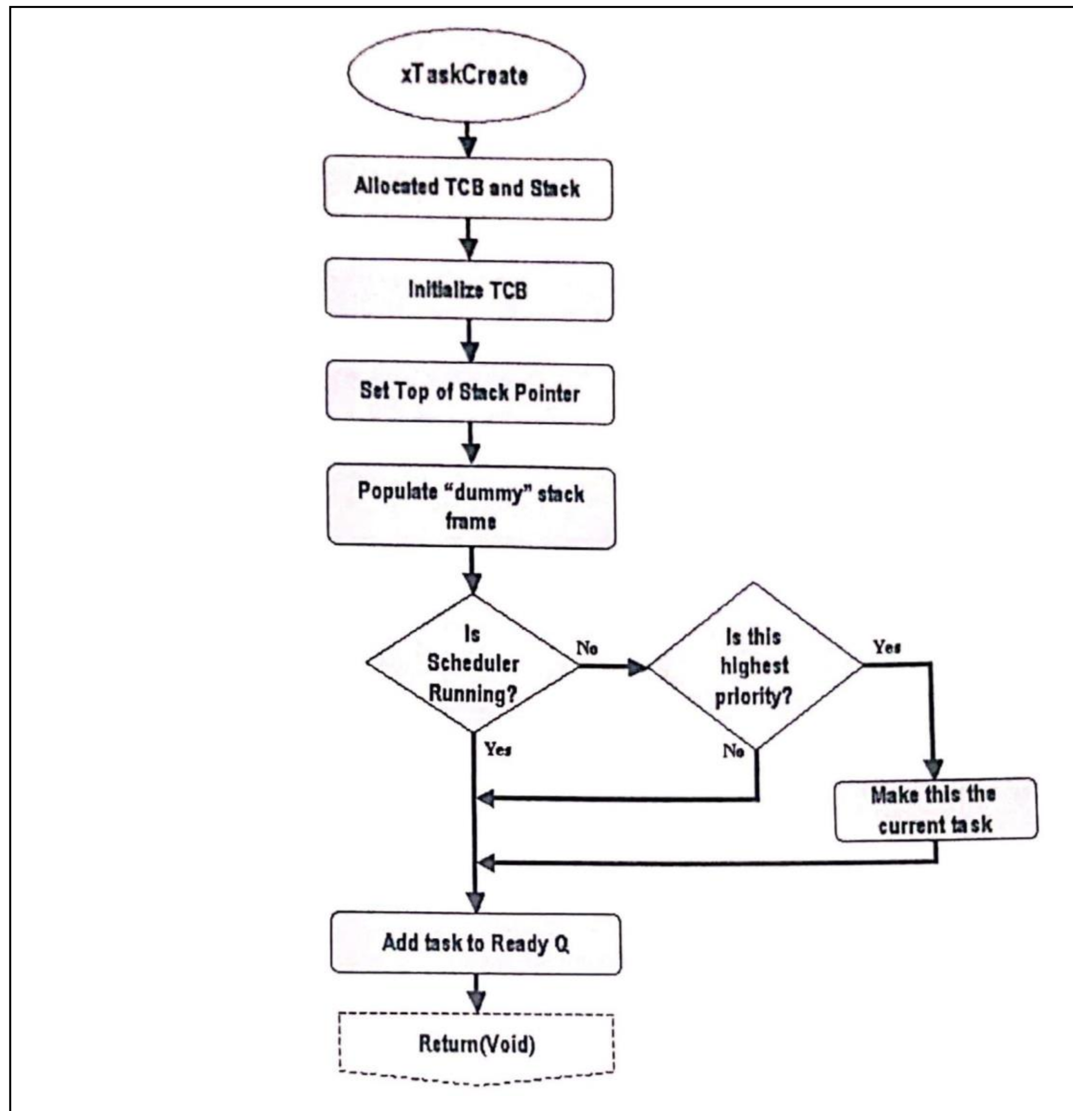



Figure III-16 : Vue de Création d'une tâche

***xTaskCreate*** doit, premièrement, allouer de la mémoire pour le TCB et la pile de la tâche. Ceci est accompli en invoquant ***AllocatoTCBandStack*** montré sur la figure III-17. Cette fonction invoque ***portMalloc*** pour obtenir un block mémoire pour le TCB qui à la taille de la structure TCB et un block mémoire de taille équivalente à la taille du type de données de la pile (ex., 8, 16 bits) multiplié par la taille de la pile requise. Les deux blocks mémoire sont obtenus à partir du heap dont la taille maximale est spécifiée dans le paramètre configTOTAL_HEAP_SIZE. Comme dernier exercice, ***allocatoTCBandStack*** définit un pointeur vers l'adresse de base de la pile à l'interieur du TCB.

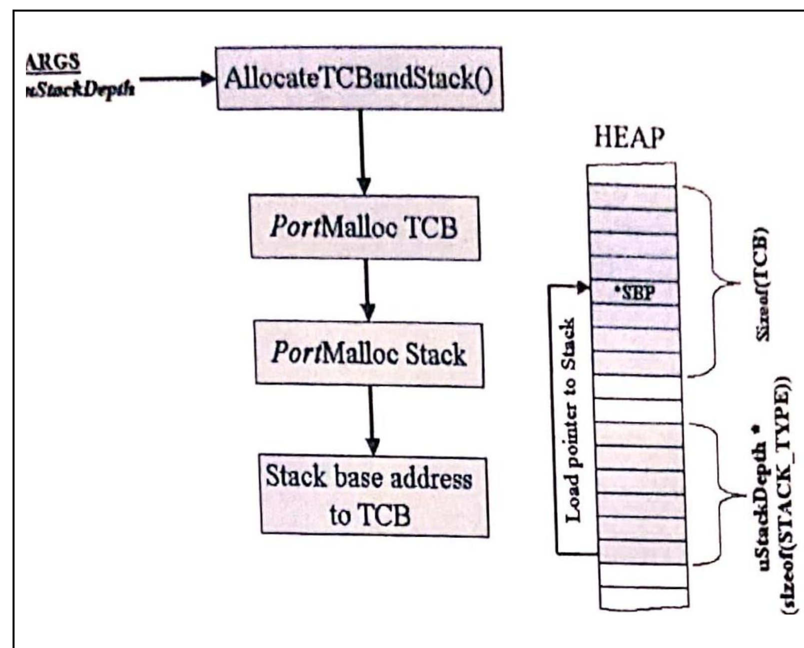



Figure III-17: Allocate Stack and TCB Memory

*portMalloc* est implémenté au sein du HAL (Hardware Abstraction Layer). Si on choisit d'une manière spécifique de compiler une des trois méthodes d'allocation mémoire *heap1.c*, *heap2.c*, 'correspondante *portMalloc* )peut être atteinte. Par exemple, *heap1.c,* implémente une politique d'allocation mémoire heap à une tâche une fois pour toute et ne permet pas la libération de cette mémoire, Cette politique est appropriée pour une application ayant un ensemble connu de tâches et qui ne varie pas avec le temps. La politique de *heap2.c* permet l'allocation et la libération de mémoire en utilisant l'algorithme Best-Fit pour localiser le block demandé, mais ne permet pas la combinaison de blocks libres adjacents en un seul block de taille plus grande. La *heap3.c* fournit simplement des enveloppes (wrappers) pour les fonctions traditionnelles *malloc[]*, and *Free[].*

Si on se réfère à la Figure III-16, la deuxième tâche exécutée par *xTaskCreate* (en assumant que la mémoire a été obtenu avec succès) est d'initialiser le TCB avec des valeurs connues, tels que le nom de la tâche, sa priorité, la profondeur de la pile à partir des paramètres d'appel de la fonction *xTaskCreate*.

La troisième et quatrième étape de *xTaskCreate* prépare la tâche pour sa première commutation de contexte. Un pointeur vers le top de la pile est initialisé avec l'adresse de base trouvé dans le TCB de la tâche (un ajustement est nécessaire qui dépend du mécanisme de variation du pointeur de pile de la cible- certaines cibles font diminuer le stack pointer d'autres font l'opposé). Ensuite la pile de la tâche est initialisée avec des valeurs

insignifiantes d'une manière telle que cela corresponde à ce qui est demandé quand une commutation de contexte est effectuée par portRESTORE_CONTEXT comme décrit ci-dessous.

Le contenu du cadre de contexte est montré à la Figure III-18. L'élément important du cadre de contexte est l'adresse de retour qui pointeur vers l'adresse de début du code de la tâche.

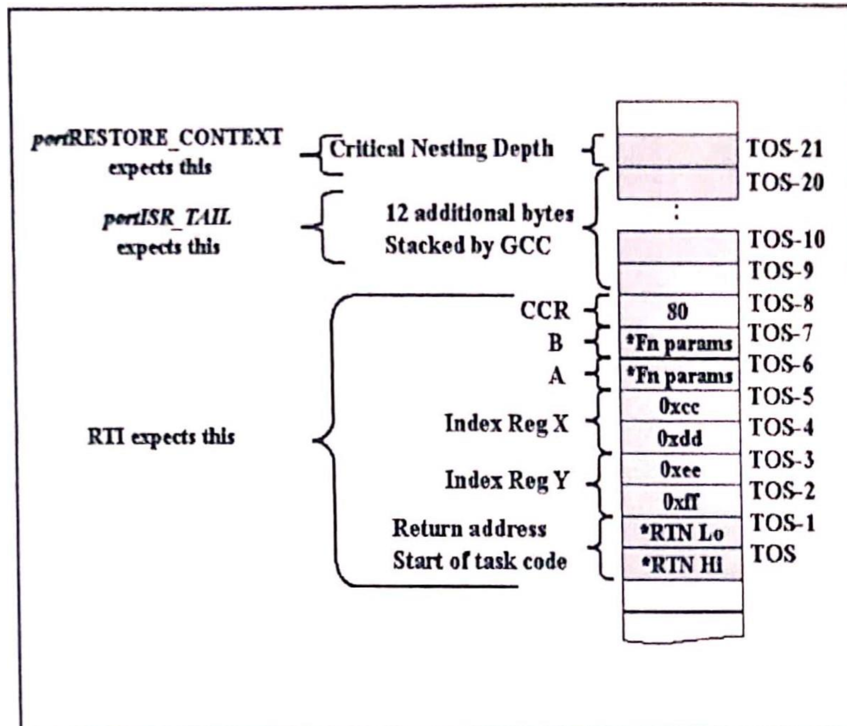



Figure III-18: Dummy Stack Frame

Après avoir initialisé la pile de la tâche, le pointeur de la pile est mémorisé (sauvegardé) dans le TCB. Cette valeur de pointeur de pile qui pointe vers le dernier élément empilé est extraite directement pour effectuer une commutation de contexte.

Chaque fois qu'une tâche est créée avec succès, xTaskCreate doit déterminer si le Scheduler est en exécution (a été démarré). Si est le cas, alors la nouvelle tâche créée est tout simplement ajoutée à la liste des prêtes (ReadyList) et le Scheduler, à sa première ou à la suivante interruption du tick d'horloge, déterminera quelle est la tâche la plus prioritaire. Si le Scheduler n'est pas en exécution (non encore démarré), alors xTaskCreate doit déterminer si la tâche qui vient d'être créée est la plus prioritaire et alors il doit faire la trace (tracker) de cette tâche en utilisant la variable globale pxCurrentTCB.

La dernière étape de la création de la tâche est d'ajouter dans la liste des prêtes (ReadyList). Ceci est fait grâce à la fonction prvAddTaskToReadyQueue . Cette fonction détermine le niveau de priorité de la tâche et puits l'insérer dans la liste des prêtes. Si la liste n'existe pas (niveau de priorité non encore créé) dans le cas où la tâche est créée dynamiquement après le

démarrage de l'application alors la liste correspondante sera créée. pxCurrentTCB est ajusté par *prvAddTaskToReadyQueue* pour tracker le TCB afin qu'il soit commutable la prochaine fois.

### III.2.5.3. Commutation entre les tâches

Le Scheduler répond à l'ISR de l'horloge. Une deuxième ISR est nécessaire pour relancer une tâche dans le cas où la tâche a été bloquée ou s'est terminée l'étape précédente. Ceci est implémenté par une SWI (Interruption Software). Tout appel à myocausera l'instruction de langage assembleur SWI à s'exécuter qui, invoquera le code ISR associé à cette interruption (défini dans port.c comme étant m'ttW) L'instruction SWI construit un cadre de contexte comme décrit précédemment— elle exécute *portISR_HEAD* et *portSAVE_CONTEXT*, détermine si une commutation est requise et charge l'entête du TCB de la nouvelle tâche dans le stack pointer si est nécessaire, et ensuite dépile le cadre du contexte d'une manière appropriée. Il faut noter que l'instruction « SWI » est non masquable contrairement à l'interruption du timer du Scheduler qui peut être masquée.

### III.2.5.4. Démarrage du Scheduler

La Figure III-19 montre les opérations qui se découlent quand le Scheduler est démarré, un appel à la fonction *vTaskStartScheduler()* du noyau FreeRTOS, doit être le dernier appel à une fonction, fait dans la fonction *main.c* après que toutes les tâches requises soient créées par l'utilisation de la fonction *xTaskCreate()*

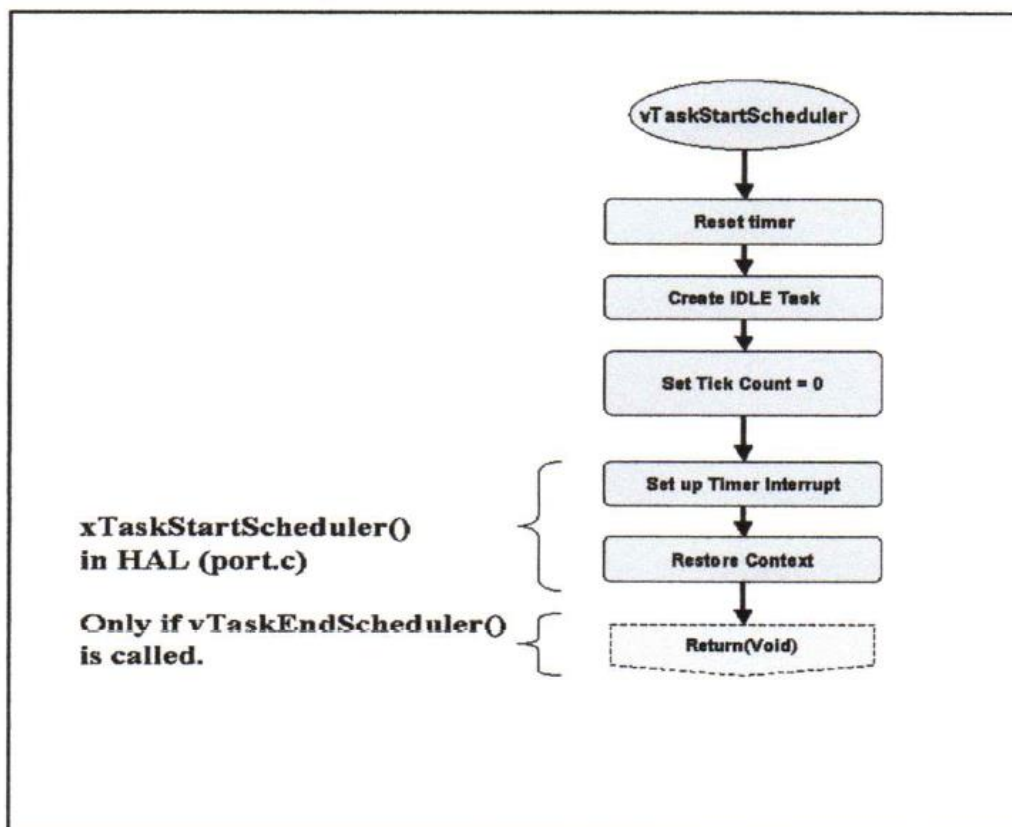



Figure III-19: FreeRTOS Task Scheduler Startup

La fonction *vTaskStartScheduler()*, premièrement , crée la tâche IDLE (processus inactif) avec la plus basse priorité et initialise la variable globale xTickCount relative au compteur à zéro. La variable globale xSchedulerRunning est mise à TRUE (vrai). Cette variable est utilisée dans plusieurs solutions pour déterminer si le Scheduler est disponible pour prendre des décisions relatives au Sheduler ou bien si ces dites décisions doivent être faites localement. Par exemple, les tâches peuvent être créées avant et après le démarrage du Scheduler. Quand les tâches sont créées avant le mécanisme de création inclue une méthode pour déterminer si la tâche qui vient d'être créée est une tâche avec la nouvelle priorité commute pxCurrentTCB pour refléter cet état (sans faire de commutation du contexte), Sinon, le Sceduler est utilisé.

*vTaskStartScheduler()* passe le contrôle à *xTaskStartScheduler()* dans la couche d'abstraction du hardware (HAL). Le HAL est requis à ce point car la première action à faire de *xTaskStartScheduler()* dans est de configurer l'interruption du timer pour invoquer le Scheduler. Puisque le timer est dépendant du hardware, sa configuration doit être faite au niveau du HAL.

La dernière chose que fait *xTaskStartScheduler()* dans est de restaurer le contexte de la tâche actuelle sélectionnée qui est pointée par pxCurrentTCB et qui est, par conséquent, la tâche la plus prioritaire. Le contexte est commuté par l'appel de ***portRESTORE_CONTEXT*** et ***portISR_TAIL***. Ceci peut paraitre come une erreur logique puisque aucune tâche n'était en exécution, cependant, comme décrit précédemment, chaque tâche possède un cadre de contexte avec des valeurs non significatives quand elle est créée. Ce cadre de contexte fournit l'adresse de départ et l'entête du TCB de la tâche est un pointeur vers la pile de la tâche. Ceci représente toutes les informations pour démarrer la tâche.

### III.2.5.5. Suspension du Scheduler

FreeRTOS offre à une tâche la possibilité de monopoliser le processeur aux dépens des autres tâches pour une durée illimitée de temps en suspendant le Scheduler. Certainement cette possibilité est utilisée par FreeRTOS lui-même. Une tâche peut concevoir de suspendre le Scheduler au cas où elle veut faire du traitement pour de longue période de temps sans rater aucune interruption. L'utilisation des sections critiques bloque toutes les interruptions y inclus les interruptions du timer. Etendre une section critique au delas des limites nécessaires viole le principe de maintenir les sections critiques courtes et brèves en temps et en espace.

Indépendamment, les opérations normales du Scheduler peuvent être suspendues à travers l'appel de *vTaskSuspendAll* et yrgstffrstrrfl. *vTaskSuspendAll* garantit que la tâche courante ne sera pas préemptée tout en continuant à traiter les interruptions (y inclues celles du timer). Les opérations normales du Scheduler reprennent par appel de *vTaskResumeAll*
Les suspensions du scheduler peuvent être imbriquées. Le niveau d'imbrication est mémorisé par la variable globale uxSchedulerSuspended dans le fichier tasks.c. La Figure III-20 montre l'algorithme implémenté pour *vTaskSuspendAll* et *vTaskResumeAll*.

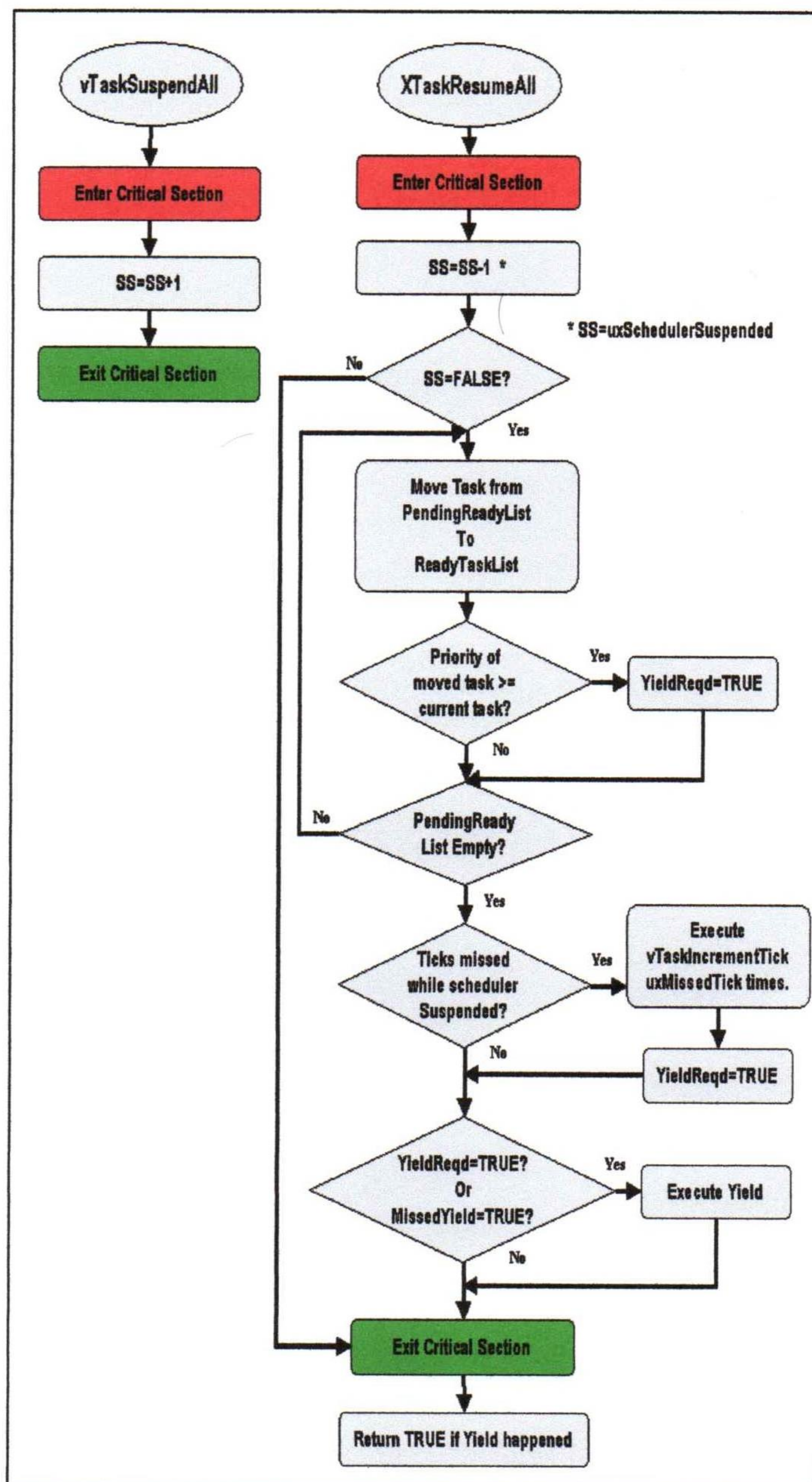



Figure III-20: Algorithms for ***vTaskSuspend*** and ***xTaskResumeAll***

Chaque fois que ***vTaskSuspend*** est exécutée uxSchedulerSuspended, est incrémentée A chaque exécution de ***xTaskResumeAll, uxSchedulerSuspended*** est décrémentée. Si uxSchedulerSuspended est different de zero (FALSE) quand ***xTaskResumeAll*** est exécutée alor rien n'est fait dans cette fonction.

Cependant, si uxSchedulerSuspended est tero (FALSE) par ***xTaskResumeAll***,alors toutes les taches qui ont été placées dans la liste (PendingReadyList) sont placées vers ta liste (ReadyTasksList}.

Une petite digression est nécessaire pour comprendre le concept general de la liste (PendingReadyList). Quand le Scheduler est suspendu, les tâches qui sont dans la listes des retardées (Delayed) ou dans a liste d'attente des événements, ne sont pas vérifiées à chaque tick du timer pour voir si elles doivent être réveillées. Cependant, en suspendant le Scheduler ceci ne stoppe pas les ISRs d'être exécutées et ceci peut causer des événements à débloquer des tâches, Cependant, quand le Sceduler est suspendu, les ISR ne peuvent pas modifier la liste des prêtes (Ready). Par conséquent, les tâches qui sont réveillées, suite à une ISR, sont placées dans la liste {PendingReadyList) et sont servies par le Scheduler quand il n'est plus suspendu.

Dans ***xTaskResumeAll*** comme chaque tâche de la liste {PendingReadyList) est déplacée dans la liste {ReadyTasksList) la priorité e cette tâche est comparée à celle de la tâche actuelle e exécution, si elle est plus grande, alors une commutation est requise dès que c'est possible pour donner le contrôle à la tâche la plus prioritaire. Il faut noter qu'il puisse exister plusieurs tâches avec une priorité supérieur à celle de la tâche courante- la commutation doit déterminer quelle est à tâche la plus prioritaire pour commuter vers cette tâche, sinon la tâche courante s'exécutera jusqu'au prochain tick.

Si on ratait des ticks du timer pendant que le Scheduler était suspendu, ceci est visible dans la variable globale uxMissedTicks. ***xTaskResumeAll*** va tenter pour se rattraper sur ces ticks en exécutant ***vTaskIncrementTIck*** une seule fois (une fois pour chaque uxMissedTicks). Si des ticks ratés existent et sont traités, ils peuvent réveiller des tâches avec des priorités supérieur à celle de la tâche courante. Par conséquent, une commutation est nécessaire dès que c'est faisable.

L'algorithme est montré sur la Figure III-21. Cette fonction est appelée une fois à chaque tick de la couche HAL (à chaque fois que la routine ISR du timer se produit). La branche droite de l'algorithme s'occupe des opérations normales du Scheduler alors que celle de gauche s'exécute quand le Scheduler est suspendu. Comme décrit ci-dessus, la branche droite simplement incrémente le compte des ticks et ensuite vérifie si l'horloge à fait un dépassement (overflow). Si c'est le cas alors les pointeurs vers les listes {DelayedTask) et {OverflowDelayedTask} sont échangés et une valeur compteur globale fait le tracking du nombre de dépassement est incrémentée. Une incrémentation du compte des ticks peut causer le réveil d'une tâche retardée (Delayed) alors une verification est faite.

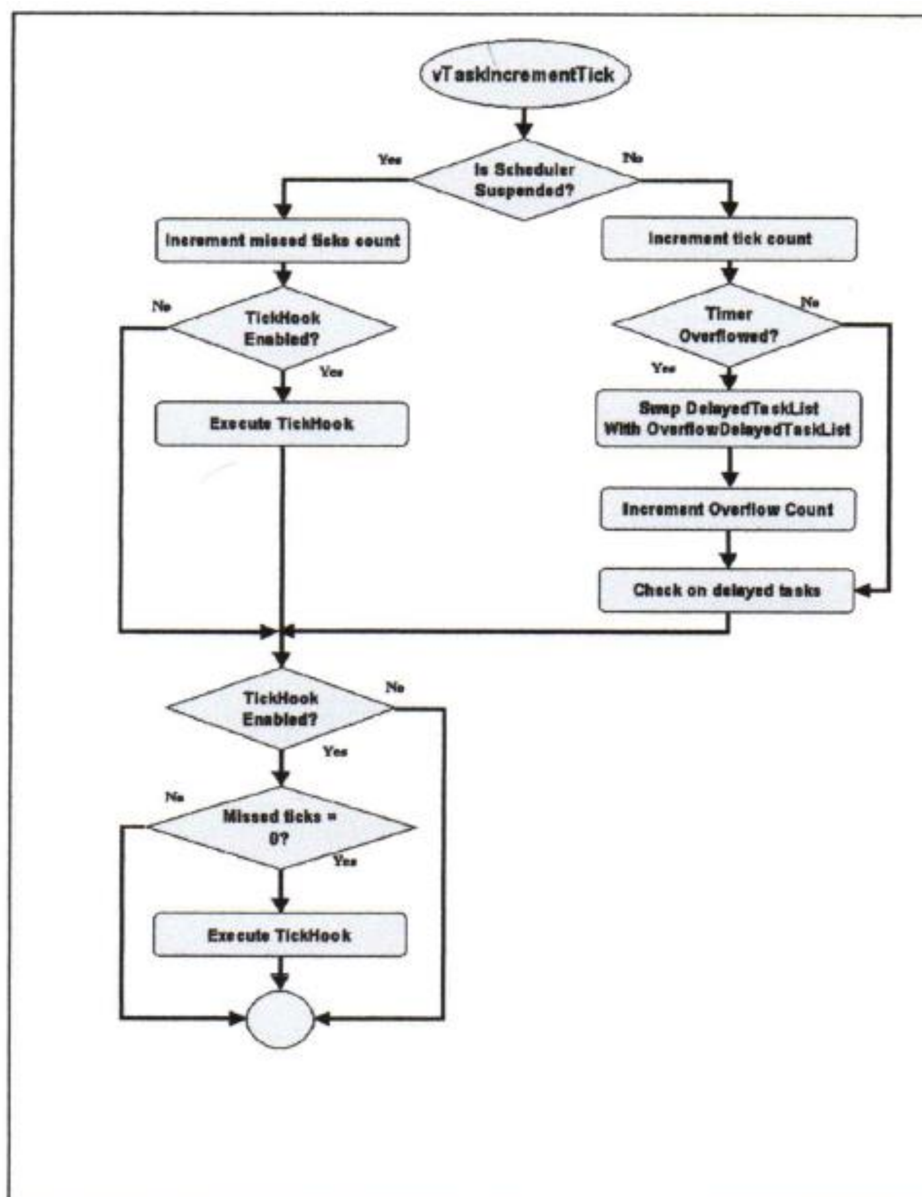



Figure III-21: Algorithm for ***vTaskIncrementTick***

Si le Scheduler est suspendu, alors la valeur globale du compte des ticks ratés est incrémentée. Si les crochets liés au ticks (tick hook) sont valides dans ***FreeRTOSConfig.h*** alors toute fonction hook est exécutée- il faut noter que les ticks Hooks fonctionent que le Scheduler soit suspendu ou non.

La section finale de l'algorithme encore vérifie si les ticks Hooks ont été valides. Une seconde vérification est faite pour voir s'il n'y a pas des ticks qui ont été ratés. Si les deux conditions sont vraies alors la fonction Hook est exécutée. Cette section finale garantie que les ticks Hooks sont exécutés à chaque fois ***vTaskIncrementTick*** est exécuté et que le Scheduler n'est pas suspendu- sauf dans le cas où les comptes des ticks sont traités une seule fois pour réduire le compte des ticks ratés à zéro (par exemple, comme discuté dans la Figure III-16 pour **xTaskResumeAll**). Dans ce cas, les ticks Hooks seront exécutés une seule fois pour l'ensemble des ticks ratés.

### III.2.5.6. Vérification de la liste des tâches retardées (Delayed Task list)

Le Scheduler vérifie la liste {DelayedTaskList) à chaque tick et localise toute tâche dont la valeur absolue de son temps est inférieure à la date courante. Les tâches sont déplacées dans la liste des prêtes (ReadyList). Les tâches retardées sont placées dans la liste {DelayedTaskList} dans l'ordre de leur valeur absolue de la date de réveil. Par conséquent la vérification se termine quand la première tâche retardée ayant une date non expirée est trouvée.

### III.2.5.7. Traitement de la section critique (Critical Section Processing)

FreeRTOS implémente les sections critiques par le masquage des interruptions. Les sections critiques sont invoquées à travers *taskENTER CRITICAL()* (qui correspondra à portENTER CRITICAL() du moment qu'entrer dans la section critique invoquera des opérations dans la couche HAL. Il existe comme équivalent la fonction *taskEXIT CRITICAL()*.

Les sections critiques peuvent être imbriquées. L'imbrication aura lieu quand une fonction entre dans une section critique pour faire un certain traitement et, tout en étant dans cette section critique fait l'appel à une fonction qui appelle une section critique (les deux fonctions peuvent être implémentées pour opérer d'une façon indépendante). Si l'imbrication n'est pas implémentée, la deuxième fonction exécutera une sortie (EXIT) de sa section critique (validant ainsi les interruptions) quand elle se termine et retourne à la première fonction qui s'attend à ce que les interruptions sont toujours masquées. En utilisant un compteur d'imbrication, chaque fonction incrémente le compte suite à l'entrée en section critique et décrémente à la sortie de la section critique. Si le compte est décrémenté et est égale à zéro, les interruptions seront validées sans aucun risque.

Dans FreeRTOS, le niveau de l'imbrication est mémorisé dans la variable uxCriticalNesting qui est empilé comme élément du contexte de la tâche. Ceci signifie que chaque tâche maintient une trace de son propre compte de niveau d'imbrication parce que c'est possible qu'une tâche puisse être commutée à partir d'une section critique

# *IV* ***Chapitre 4: FreeRTOS+TRACE***

# IV.1. FreeRTOS + Trace

## IV.1.1. Introduction

FreeRTOS + Trace est un outil de diagnostic pour les systèmes d'exploitation des systèmes embarqués basés sur FreeRTOS. FreeRTOS + Trace donne un aperçu sur le comportement d'exécution, ce qui permet de réduire le temps de débogage donc au temps de mise en œuvre de l'application. Ce qui permet d'économiser le coût d'investissement de cet outil.

FreeRTOS + Trace a été développé en partenariat avec les développeurs RTOS et les ingénieurs Real Time Ltd, Percepio développé depuis 2004.

FreeRTOS + Trace fournit plus d'une douzaine de vues sur le comportement d'exécution, y compris la planification des tâches et le Sceduler, les Interruptions, l'interaction entre les tâches, ainsi que des événements d'utilisateurs générés à partir de l'application. FreeRTOS + Trace peut étre utilisé côte à côte avec un débogueur traditionnel et complète la vue de débogage dans une perspective de niveau plus élevé, idéal pour comprendre les erreurs complexes où la perspective des débogueurs classiques est trop étroite.

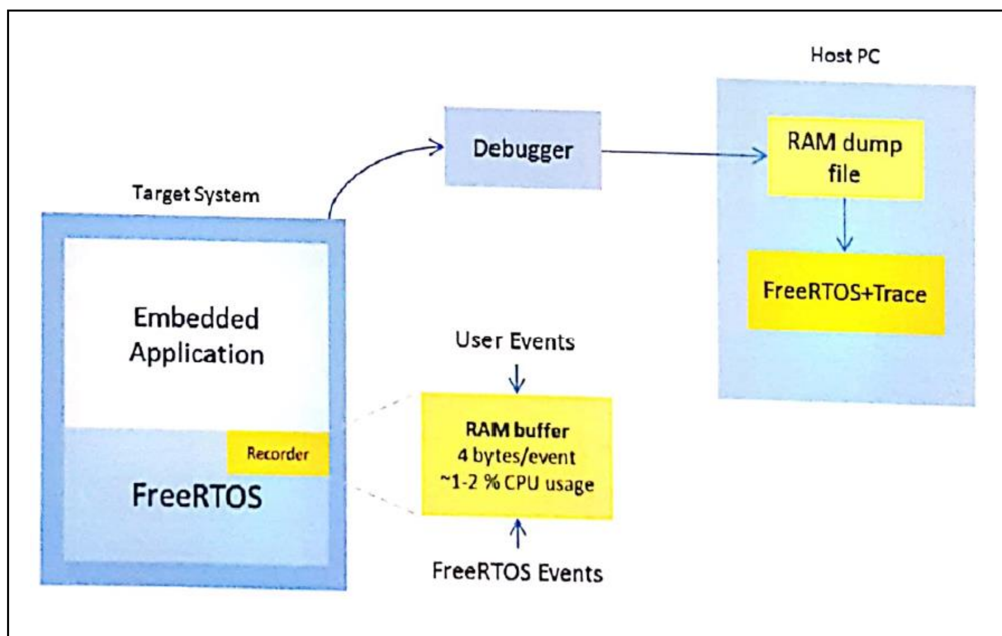
freeRTOS trace est plus qu'un simple observateur. Il contient plusieurs analyses avancées développées depuis 2004, il permet rapidement de comprendre les données de trace. Par exemple Il relie tes événements. Ce qui vous permet de suivre les messages entre les tâches et de trouver l'événement qui déclenche une instance de tâche particulière. En plus, il offre diver ponts de we de plus haut niveau tels que le graphe de flux de communication et le graphique de la charge du processeur, ce qui rend plus facile de trouver des anomalies dans une trace.

FreeRTOS + Trace ne dépend pas de matériel de trace supplémentaire, ce qui signifie qu'il peut être utilisé dans les systèmes déployés pour trouver les erreurs rares ou indécelables par les outils classiques de trace ou de débogage.

### IV.1.2. Présentation du système

Le FreeRTOS + solution Trace se compose de deux parties:

- L'application PC (FreeRTOS + Trace), qui visualiser des fichiers de données d'enregistrement.
- Une bibliothèque de l'enregistreur de trace qui s'intègre avec FreeRTOS, prévu dans le code source C.




L'application PC FreeRTOS + trace a été développée pour Microsoft Windows. Versions de FreeRTOS + trace pour les autres systèmes d'exploitation peuvent être fournis sur demande.

La bibliothèque de l'enregistreur trace stocke les données d'événement dans un tampon de RAM, qui est envoyé sur demande à l'ordinateur hôte en utilisant la connexion débogueur existante. La bibliothèque de l'enregistreur est rapide, configurable .Son tampon de RAM peut être réduit à n'utiliser que quelques Ko de RAM, ce qui permet une utilisation dans les systèmes microcontrôleurs avec la RAM sur puce seulement.

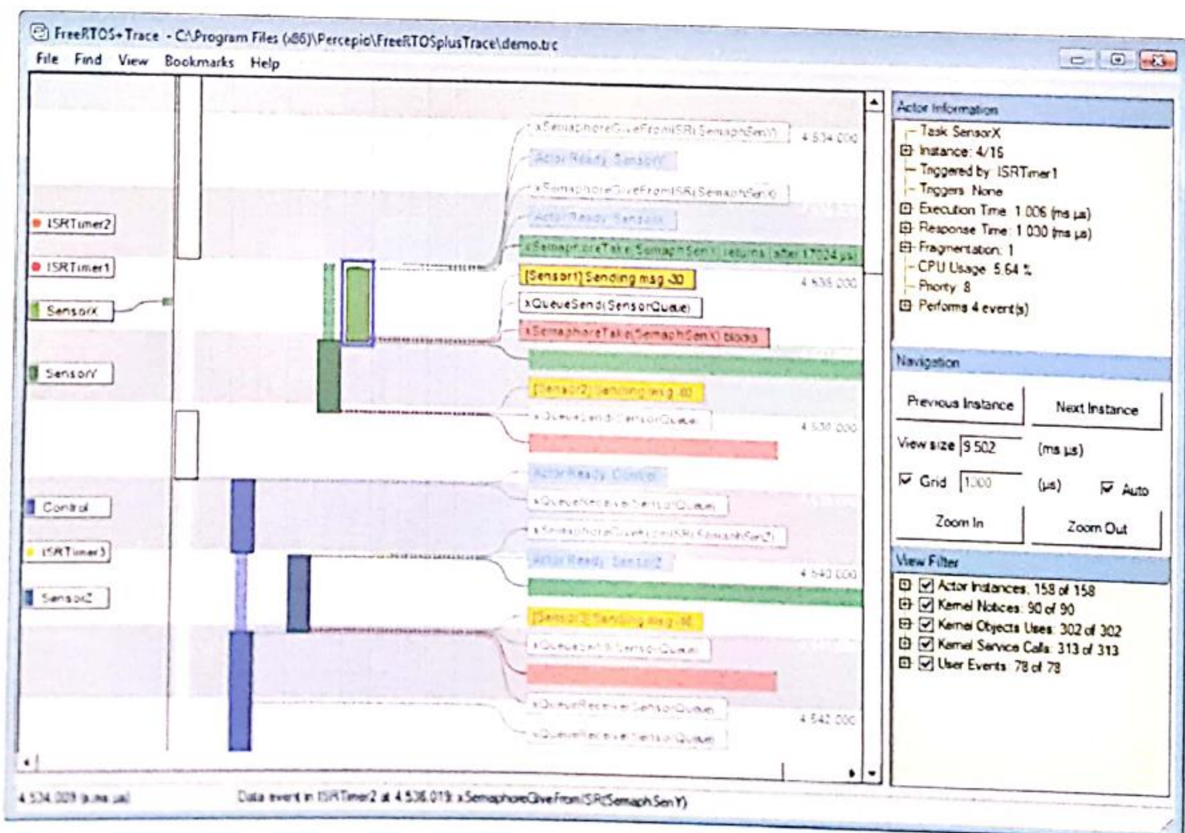
La bibliothèque de l'enregistreur de trace utilise seulement quatre (4) octets par événement, dans la plupart des cas, tandis que d'autres solutions de traçage utilisent généralement 8 ou même 16 octets par événement. Avec FreeRTOS + Trace, une mémoire RAM de 64 Ko donne ainsi jusqu'à 16.000 événements histoire de trace. Un tampon 4 Ko (1000 événements) est généralement suffisant pour étudier un scénario particulier, et obtenir un aperçu de l'activité périodique, ou d'étudier la mise en service du système.

Les traces sont téléchargées sur le PC hôte. Cela se fait en stockant le contenu de la RAM dans un fichier hôte-PC, qui est ensuite ouvert en FreeRTOS + Trace. Il est également possible de stocker les données de suivi de la RAM vers un système de fichiers sur l'appareil, s'il est disponible, ou de transférer les données via une interface existante, comme une connexion USB. Cela exige cependant la mise en oeuvre d'un mécanisme de téléchargement

### IV.1.3. Utilisation FreeRTOS + Trace

FreeRTOS + Trace fournit plusieurs vues graphiques qui donne des perspectives différentes sur le comportement d'exécution, sur la base d'un enregistrement de la planification des tâches et des appels de service du noyau FreeRTOS. On peut également choisir d'inclure des routines de service d'interruption (ISR) ainsi que des événements d'application en ajoutant des appels propres à la bibliothèque de l'enregistreur de trace.

Cette section a pour but de donner un aperçu rapide des fonctionnalités et comment cet outil est utilisé.

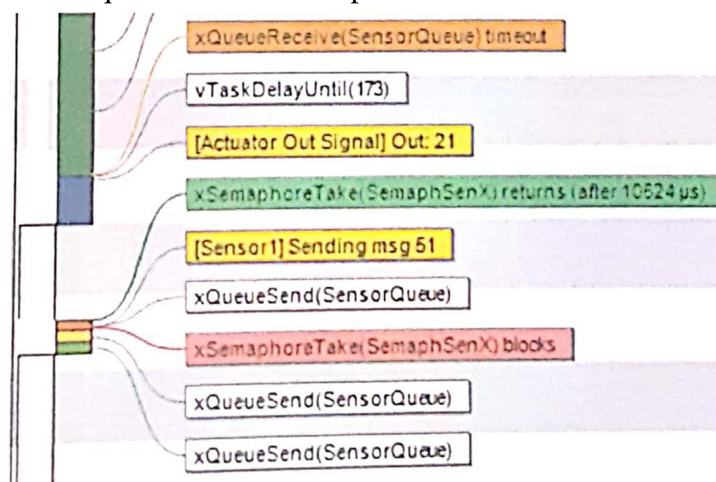


**Le point de vue trace principale** fournit toutes les informations enregistrées sur une ligne de temps verticale. Ce point de vue est complété par plus d'une douzaine d'autres vues fournissent des aperçus de haut niveau ou des vues centrées sur des points de vue différents. L'ordonnancement des tâches est présenté en utilisant des rectangles codés par couleur, où la couleur contribue à identifier l'acteur. Par acteur , on entend un thread d'exécution - une tâche ou d'interruption. Les couleurs acteurs sont tirées d'un schéma de couleur en fonction de la priorité acteur. La palette de couleurs par défaut est le spectre de la lumière naturelle, allant du rouge (haute priorité) au bleu (basse priorité), et avec des tons plus clairs de bleu pour les tâches les moins prioritaires et, enfin, blancs pour la tâche inactive. Les couleurs exactes utilisées dépendent du nombre d'acteurs dans la trace.

L'exécution des tâches et interruptions peuvent être visualisées en utilisant trois modes de visualisation:

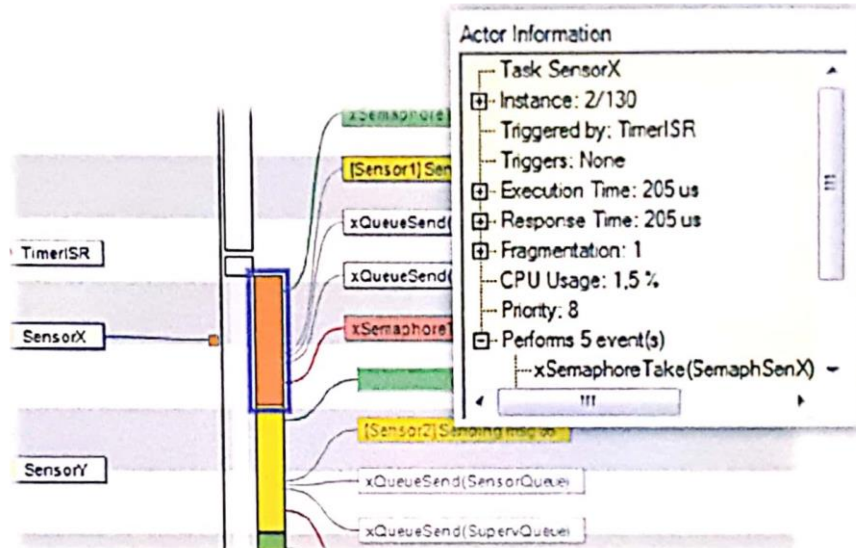
- **Mode d'affichage de Gantt :** Affiche une colonne par tâche et d'interruption. Ces mieux pour repérer les tâches rares ou terminées. C'est le mode de visualisation par défaut. (Touche de raccourci "G")
- **Modes de vue fusionnée :** Affiche toutes les tâches et les interruptions dans une seule colonne, avec tirets **sideway** pour montrer une préemption et un blocage. Cela donne le meilleur sens de l'ordre d'exécution. (Touche de raccourci "M')
- **Diviser Modes de vue :** Présente les tâches et les interruptions en deux colonnes, avec les tirets comme dans le mode de vue fusionnée. Cela supprime le «bruit» interrompt en les présentant séparément. (Touche de raccourci "S")

Les événements du noyau tels que les appels aux services du noyau et les événements utilisateur sont affichés comme des étiquettes de texte sur le côté droit de la trace d'ordonnancement. Les étiquettes sont codées par couleur en fonction de l'état de l'opération.




- Les indications en rouge - les appels systèmes bloquants.
- Les étiquettes vertes - le retour des appels systèmes bloquants.
- Étiquettes blanches - les appels système qui ont rempli sans blocage.
- Étiquettes Orange - appels système qui sont retournés en raison d'un délai d'attente.
- Étiquettes Jaunes - événements utilisateur.

Notez que **vTaskDelay** et **vTaskDelayUntil** sont représentées en blanc, pas rouge, mémo s' ils bloquent la tâche d'exécution. La raison en est que ceux-ci se traduisent par un blocage inconditionnel, ce qui est l'objectif visé par ces services du noyau. Les étiquettes rouges n ° sont utilisés que pour le blocage des ressources partagées, par exemple, les files de messages, les sémaphores et les mutex, ce qui est souvent non intentionnelle et donc d'un plus grand intérêt à étudier.

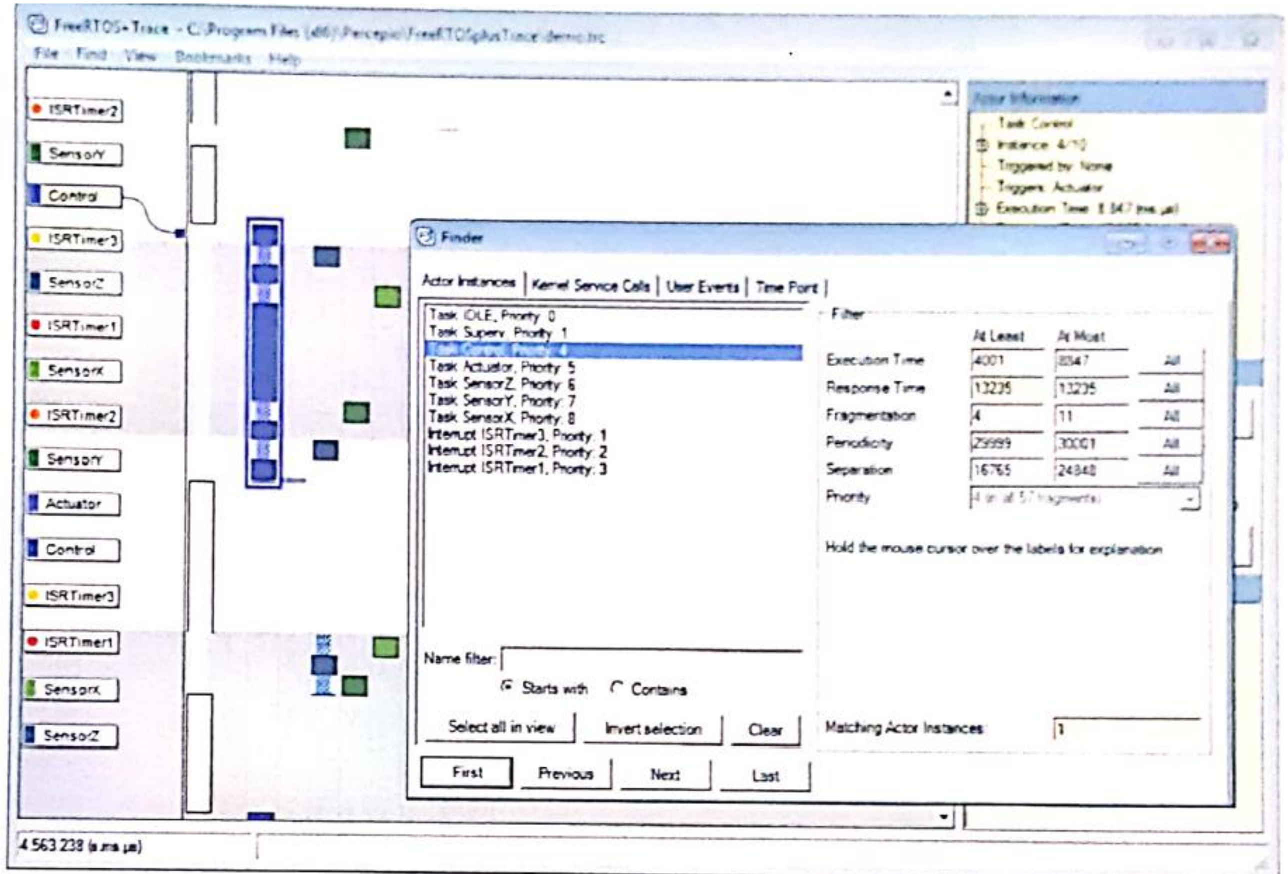



**En cliquant** sur un acteur, l'appel système ou d'un événement utilisateur affiche des informations sur l'Acteur, comme illustré ci-dessus. Il s'agit d'une structure arborescente contenant beaucoup d'informations.

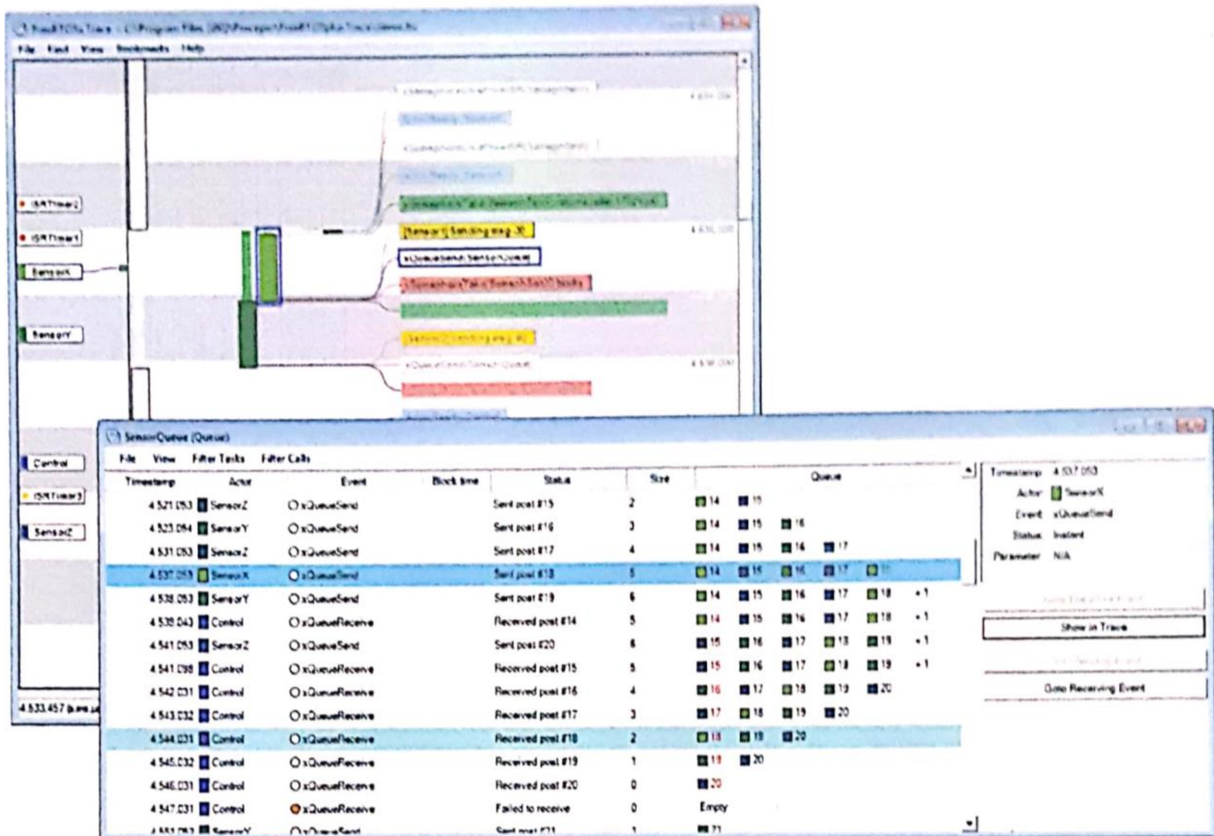
**Un double-clic** sur un acteur, l'appel système ou d'un événement utilisateur ouvre une vue ciblée affiche la liste de tous les événements liés, la vue historique de l'objet. Cela montre tous les appels système sur un objet noyau sélectionné, c'est à dire, une file de messages, sémaphore ou mutex. Double-cliquer sur un événement utilisateur affiche le journal des événements d'utilisateur et affiche la liste de tous les événements utilisateur enregistrées.

**Vues supplémentaires**

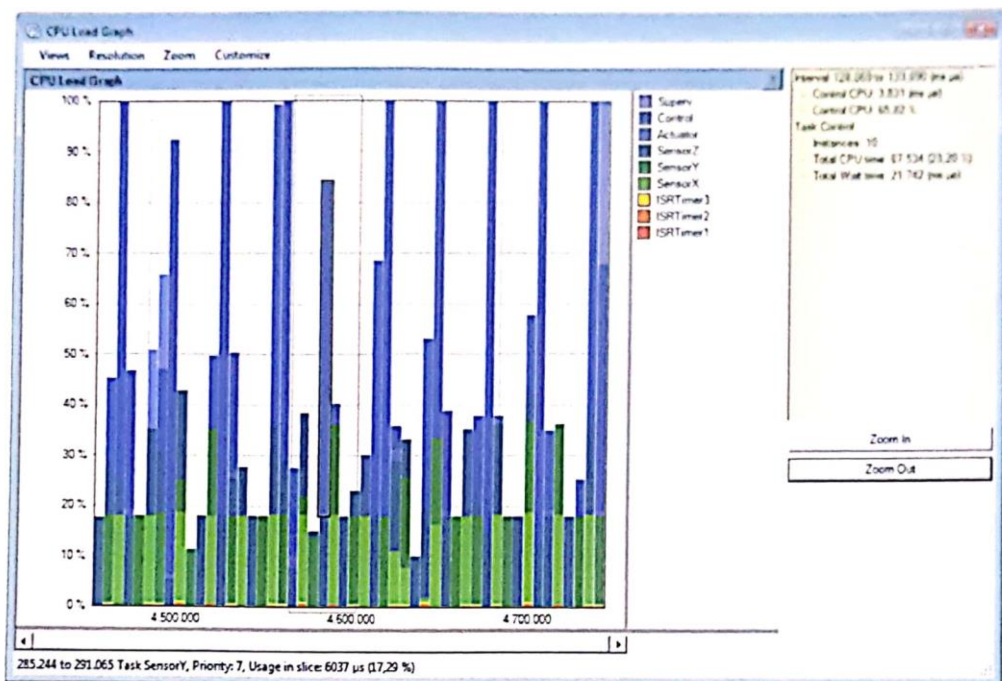
Ci-dessous sont quelques exemples des autres vues et des dialogues qui complète le point de vue de trace. Il y a plus d'une douzaine de vues au total. Toutes les vues sont connectées à la fenêtre principale de trace.




Pour trouver une tâche particulière ou interrompre la vue trace, ouvrez le **Finder** (touche de raccourci "Ctrl + F"). Dans la boîte de dialogue du Finder, on peut également spécifier des filtres sur les propriétés temporelles telles que le temps de réponse, afin de trouver rapidement les cas extrêmes.




**Afficher l'historique des objets** permet de suivre un objet noyau particulier, dans ce cas, **une file de messages**. On peut suivre un message **send** ou **receive**, ou un signal de **sémaphore**, voir la propriété du mutex. Un double-clic dans cette vue met en évidence l'événement dans la vue trace principale.



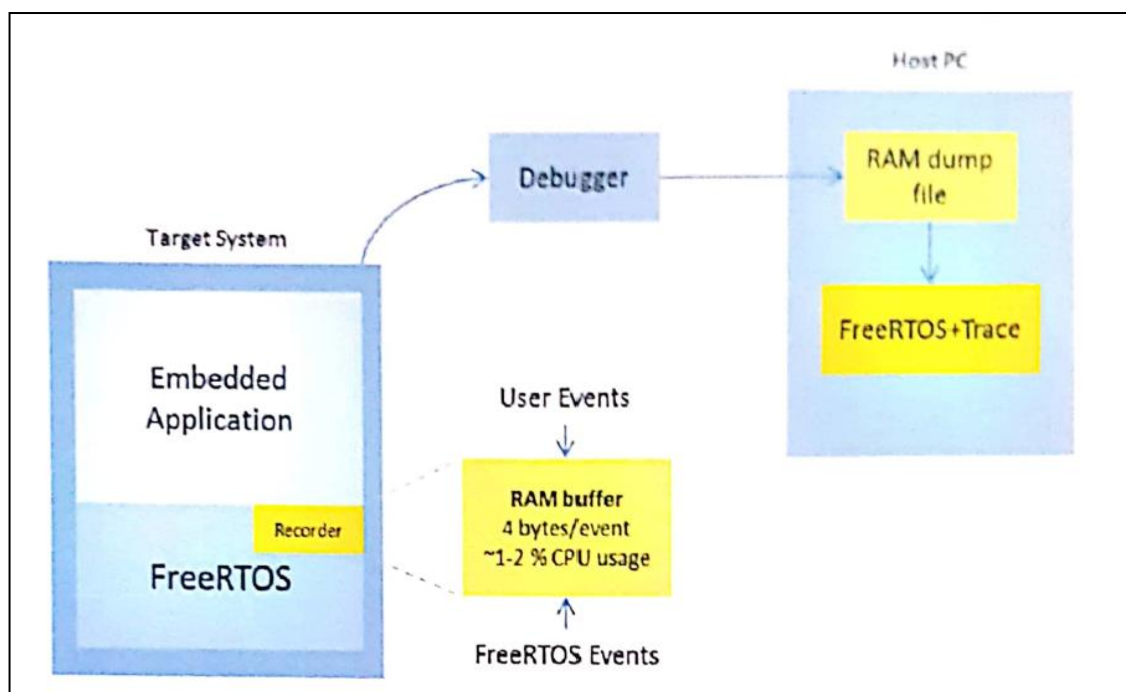

**Graphique de la charge CPU** affiche l'utilisation du processeur par tâche.

# IV.2. Enregistreur Bibliothèque

## IV.2.1. Utilisation de l'enregistreur de trace

La bibliothèque enregistreur de trace est le composant d'exécution intégré de FreeRTOS + Trace. La bibliothèque enregistreuse est intégrée dans FreeRTOS et stocke les données d'événements dans une mémoire tampon de RAM, ce qui est téléchargé sur demande à l'ordinateur hôte. Les traces sont généralement téléchargées sur le PC hôte en prenant un buffer RAM en utilisant le débogueur. C'est une caractéristique très courante dans la plupart des débogueurs, et généralement appelé "Sauvegarde mémoire" ou similaire. FreeRTOS + Trace ouvre les fichiers de données de trace au format binaire (. Bin) et au format Intel Hex (. Hex). FreeRTOS + localise automatiquement les données de trace en raison d'une signature numérique spéciale dans le début et la fin des données de trace.

L'enregistrement possède deux modes de fonctionnement.




- Arrêtez-lorsque le mode full
- Ring-tampon en mode

Le mode d'arrêt-full de l'enregistreur s'arrête quand le tampon est plein. Cela convient lorsque vous souhaitez capturer les événements suivants un point particulier de votre code où vous appelez uiTraceStart () pour lancer l'enregistreur. Le mode d'anneau tampon vous permet d'avoir l'enregistreur actif en permanence, car il écrase les événements les plus anciens avec de nouveaux événements et ainsi conserve toujours la dernière histoire jusqu'au moment où vTraceStop () est appelée pour arrêter l'enregistreur, par exemple, sur un détectée état d'erreur, ou jusqu'à ce que le système est arrêté par le débogueur.

La bibliothèque enregistreuse est dotée de plusieurs réglages qui vous devriez inspecter avant de l'utiliser, comme les deux modes décrits ci-dessus. Les réglages se trouvent dans le fichier d'en-tête **trcConfig.h** , ainsi qu'une documentation détaillée.

Pour plus d'informations sur la bibliothèque de l'enregistreur, reportez-vous **trcUser.h** (et **trcUser.c** , si vous voulez étudier la mise en œuvre détaillé),

#### IV.2.1.1 Hardware Timer Porting

Développer un port horloge matérielle dans un environnement FreeRTOS, est réalisé grâce au fichier de bibliothèque enregistreur **trcHardwarePort.c** qui contient une fonction **vTracePortGetTimeStamp** que l'on attend de lire l'heure de l'horloge matérielle / compteur qui anime le tic d'horloge FreeRTOS. Cette fonction est basée sur un ensemble de macros (avec le préfixe "HWTC") qu'on définit en fonction du votre matériel spécifique trcHardwarcPort.h . il est à noter qu'il n'est généralement pas de modifier la fonction **vTracePortGetVlmeStamp** dans **trcHardwarePort.c,** seulement macros HWTC dans le fichier d'en-tête.

L'horloge matérielle / compteur utilisé qui entraîne le tic d'horloge de FreeRTOS peut être identifié dans FreeRTOS fichier **port.c** , où il est initialisé.

### IV.2.2. FreeRTOS intégration

Pour intégrer la bibliothèque de trace de l'enregistreur dans FreeRTOS , les étapes suivantes sont suggérées:

- Vérifiez que votre version est FreeRTOS v7,3.0 ou ultérieure.
- Repérez les sources de la bibliothèque de l'enregistreur et les copier dans un endroit approprié dans le répertoire de développement 'workshop'
- Inclure les sources de la bibliothèque traces (trcBase.c, trcHardwarePort.c, trcKernel.c et trcUser.c).
- Ajoutez les lignes suivantes à la fin de votre **FreeRTOSConfig.h**

```
# Define configUSE_TRACE_FACILITY 1
# Include "trcKernelHooks.h"
```

- Sélectionnez le port d'horloge matérielle droit dans trcHardwarePorts.h . Dans le cas où Vérifiez les paramètres dans trcConfig.h et modifier si nécessaire pour correspondre à votre système (par exemple, le nombre maximum de tâches, la taille de la mémoire tampon des événements, etc.)
- Appelez vTracelnitTraceData le plus tôt possible dans votre programme principal, avant tous les autres appels de l'enregistreur et avant tout appel à FreeRTOS sont faites.
- Appelez uiTraceStart au point où vous souhaitez commencer l'enregistrement, à la mise en service ou à un certain point plus tard dans le code de l'application. N'oubliez pas de vérifier la valeur de retour - une valeur de zéro indique que l'enregistreur n'a pas pu démarrer carvTraceError a été appelé. Cela signifie qu'il ya une erreur interne dans la

bibliothèque de l'enregistreur, probablement à des valeurs insuffisantes pour les constantes Nxxx dans trcConfig.h (par exemple, NTask, nQueue, etc.) Le message d'erreur est dans ce cas affiché lorsque vous ouvrez le fichier de trace, mais peut aussi être obtenu à partir de l'API enregistreur en appelant xTraceGetLastError.

- Définir la macrovTraceConsolcMcssage dans trcHardwarePort.h . Il devrait être associé à la fonction d'impression de la console de débogage (par exemple, "printf").
- En option, vous pouvez appeler vTraceStartStatusMonltor dans votre démarrage pour démarrer la tâche de surveillance de l'état de l'enregistreur. Cette fonction rend compte périodiquement l'état de l'enregistreur à l'aide impressions de la console. Ce n'est pas nécessaire.

### IV.2.3. Télécharger les données de trace

Cette section présente la description de la façon de télécharger les données de trace du système cible à FreeRTOS + Trace. Votre débogueur existant est utilisé pour télécharger les données de suivi de la puce de mémoire vive. Il s'agit d'un vidage de RAM plaine, qui se fait chaque fois que vous voulez regarder le contenu du tampon traces. Cela signifie qu'il fonctionne avec pratiquement avec n'importe quelle sonde de débogage et débogueur IDE sur le marché, depuis l'enregistrement du contenu RAM est une fonction couramment disponible.

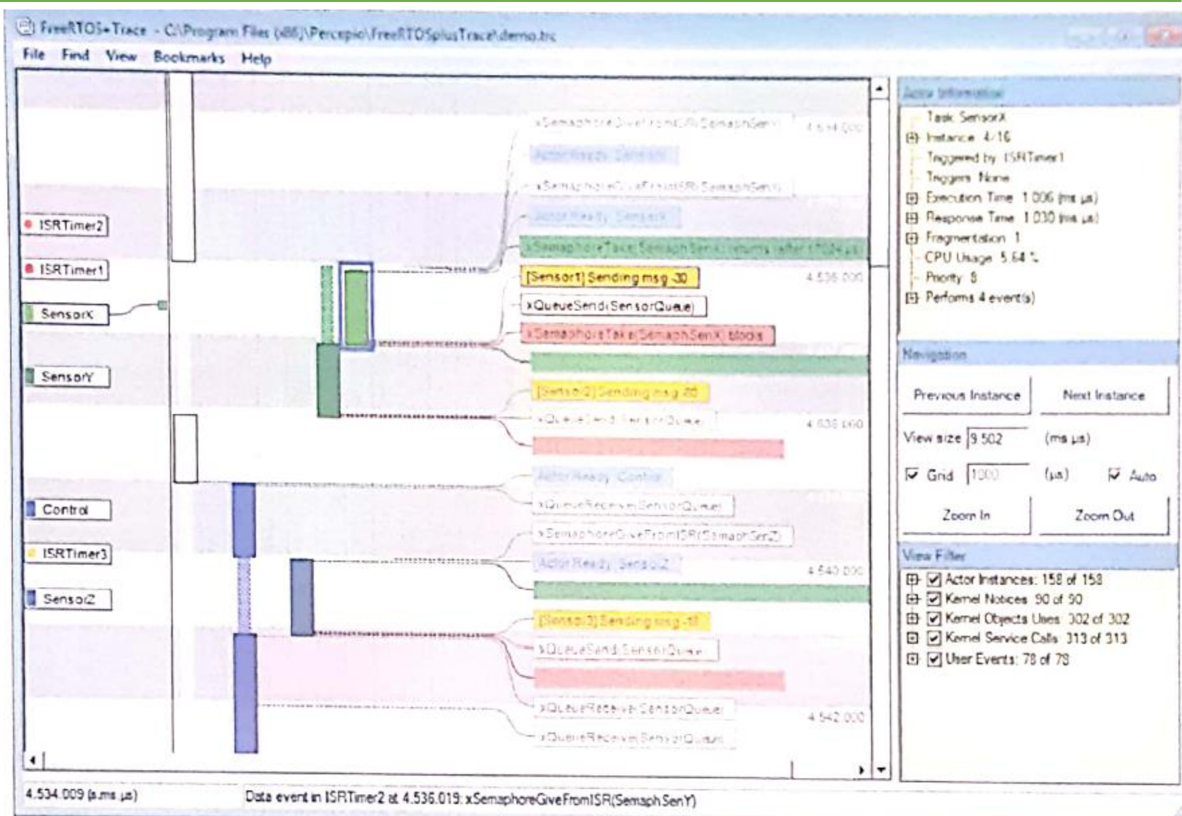
**L'utilisation d'autres environnements de développement et les sondes de débogage**
La plupart des débogueurs sont en mesure de sauvegarder le contenu de la RAM dans un fichier.FreeRTOS + Trace prend en charge les formats communs suivants:

- Binary (. Bin ou. Dump), en soutenant gdb, J-Link et Renesas HEW.
- Intel Hex (. Hex), en soutenant IAR Embedded Workbench et Atmel MemoryLogger.
- MCH (. SMI), en soutenant Microchip MPLAB.

## IV.3. Fenêtre principale

### IV.3.1. Trace View

Il s'agit de la vue principale de FreeRTOS + Trace, montrant tous les événements enregistrés sur un calendrier verticale allant vers le bas. La principale composante de ce point de vue est la planification de trace, montrant des fragments d'acteurs affichés sous forme de rectangles de couleur. Les couleurs sont uniques pour chaque acteur, et sont attribués en fonction de la priorité d'ordonnancement. Par acteur , on entend un thread d'exécution - une tâche ou d'interruption. La palette de couleurs par défaut est le spectre de la lumière naturelle, allant du rouge (haute priorité) au bleu (basse priorité), et avec des tons plus clairs bleus pour les tâches les moins prioritaires et, enfin, blancs pour la tâche inactive. Les couleurs exactes utilisées dépendent du nombre d'acteurs dans la trace.




À l'extrême gauche sont des étiquettes affichant les noms des acteurs. Lors d'un zoom sur les étiquettes de nom sont filtrés de sorte que seules les plus importantes sont affichées. Étiquettes répétées du même acteur sont ensuite masqués, en faveur d'étiquettes pour les acteurs moins fréquentes. Cependant, il y a toujours au moins une étiquette par acteur visible, et vous pouvez cliquer dans la trace pour montrer l'étiquette correspondante acteur nom.

### IV.3.1.1. Modes d'affichage

L'exécution des tâches et interruptions peuvent être visualisées en utilisant trois modes d'affichage, que vous pouvez basculer rapidement entre d'optimiser l'affichage en fonction de la situation et de problème. Les modes sont les suivants:

- **Mode d'affichage de Gantt :** Affiche une colonne par tâche et d'interruption. Cest le mode d'affichage préféré pour repérer les cas rares acteurs et pour visualiser les comportements périodiques Il s'agit du mode de visualisation par défaut. (Touche de raccourci "G").

- **Modes de vue fusionnée :** Affiche toutes les tâches et les interruptions dans une seule colonne, avec tirets sideway pour montrer préemption et de blocage. Cela donne le meilleur sens de l'ordre d'exécution, (Touche de raccourci "M").

- **Diviser Modes de vue :** Présente les tâches et les interruptions en deux colonnes, avec les tirets comme dans le mode de vue fusionnée. Cela supprime le '(bruit» interrompt en les présentant séparément. (Touche de raccourci "S").

### IV.3.1.2. Événements

L'écran principal affiche également les événements utilisateur et les appels de service du noyau, si elle est activée dans le filtre de visibilité dans le coin inférieur droit. Les événements sont préséries comme des étiquettes à code couleur à la droite du fragment dans lequel ils se produisent. Les événements peuvent être sélectionnés par clics de souris, ce qui souligne l'événement sélectionné à l'aide d'une lueur bleue. Double-cliquer sur un événement pour faire apparaître le noyau historique des objets ou le journal des événements utilisateur (en fonction du type d'événement) et il souligner l'événement cliqué.

**Notez**

Quand il y a trop d'événements dans la vue en cours pour s'adapter à l'écran, ils seront cachés et remplacés par des événements X qui ne figurent pas. Si cela se produit, effectuer un zoom avant ou utilisez le filtre de visibilité pour réduire le nombre d'événements. Les événements de l'instance acteur sélectionné sont cependant toujours affichés.

Les événements sont présentés dans des couleurs différentes selon le type et l'état. Les événements utilisateur sont affichés en jaune. Les appels de service du noyau peuvent être blanche, si l'appel est un succès et non-bloquante et rouge si l'appel bloque. Si les blocs d'appel, il y aura un autre événement où les retours d'appel (vert) ou arrive à expiration (orange). Si l'appel échoue, l'événement sera affiché dans une couleur rouge vif. Lors de la sélection d'un événement d'un appel qui bloque, l'événement correspondant, où que les retours d'appels ou arrive à expiration est également souligné. Vous pouvez accéder à cet événement en appuyant sur F8 ou en utilisant le menu contextuel.

### IV.3.1.3. Zoom et les fonctions de navigation

Pour naviguer dans le tracé, vous pouvez utiliser la molette de la souris ou la barre de défilement (à droite). Vous pouvez également faire glisser la vue en appuyant sur la molette de la souris ou le bouton central de la souris. La flèche et les boutons Page haut / bas peut également être utilisé lorsque la zone d'affichage de trace a le focus.

**Notez**

Lors de la recherche d'un endroit particulier, il est souvent plus facile d'utiliser le Finder ou l'un des aperçus graphiques pour le trouver. Lorsque vous utilisez une autre vue de naviguer sur le tracé, vous pouvez cliquer ou double-cliquer pour concentrer la vue trace à cet endroit. Toutes les vues graphiques prennent également en charge la sélection d'un intervalle de temps (en appuyant sur le bouton gauche de la souris et glisser) et montrant cet intervalle dans la vue de trace.

Pour effectuer un zoom avant ou arrière, utilisez les boutons de zoom sur le panneau d'outils, le pavé numérique boutons + et - ou les options de zoom dans le menu clic droit. Vous pouvez également zoomer avec la molette de votre souris lorsque vous tenez la touche Ctrl enfoncée. Le comportement de défilement-to-zoom est toujours actif lorsque Scroll bock est activée.

Si votre souris (lispoe.e boutone, Précédent et Suivant, voue, pouvez utiliser pour effectuer rapidement un zoom avant et arrière.

### IV.3.1.4. Outil de tableau

A droite de la vue de trace est la palette d'outils. Au sommet trouve l'affichage d'informations Acteur, qui affiche des informations sur l'acteur Sélectionné et l'instance acteur actuellement sélectionné, telles que le calendrier, la fragmentation et les événements qui se produisent. Les valeurs de propriété pour l'instance sélectionnée sont affichées, ainsi que la moyenne acteur, maximale et minimale. Double-cliquez sur la valeur maximale ou minimale Indique l'instance correspondante dans la trace.

Ci-dessous l'affichage d'informations Acteur y a des boutons pour naviguer vers l'instance précédente et suivante du même acteur,

Vient ensuite les paramètres d'affichage. Vous pouvez régler la taille d'affichage (niveau de zoom) en tapant la taille d'affichage souhaité dans la zone de texte vue taille et en appuyant sur Entrée. Dans le cadre du réglage de taille d'affichage sont les paramètres de la grille. La grille est les rayures blanches et grises dans la vue de trace, ce qui montre l'échelle de temps.

Ci-dessous les paramètres d'affichage des boutons pour effectuer un zoom avant et arrière.

Au bas de la palette d'outils est le boîtier de filtre qui contrôle la visibilité ce qui est affiché dans la vue de trace. Par défaut, tous les acteurs sont des événements visibles mais non. Décochant un acteur ne sera pas le cacher complètement, mais parce que c'est fragments à tirer comme indiqué rectangles gris. Aucun événement acteurs non vérifiées ne sera caché.

Pour afficher les événements de service du noyau, sélectionnez un ou plusieurs services du noyau ou des objets du noyau que vous êtes intéressé po Si vous sélectionnez des objets du noyau deux et services du noyau, les événements Inclus doit correspondre à la fois un service noyau sélectionné et un objet noyau sélectionné.

### IV.3.1.5. Les fonctions avancées

freeRTOS + Trace peut identifier les causes d'activation pour les instances acteur. Par exemple, une instance peut être activé à un autre acteur envoie à une file d'attente ou signale un sémaphore. Pour aller à la cause activation de l'instance acteur sélectionné, appuyez sur f 9.

FteeRTOS Trace peut également identifier les cas déclenchés par l'instance sélectionnée, soit à partir de l'écran d'informations Acteur («triggers») ou en appuyant sur FIO. En utilisant FIO et il y a plusieurs Instances activées par l'instance sélectionnée, vous obtenez un menu où vous pouvez choisir quel acteur actif que vous souhaitez suivre.

Pour les appels de service du noyau, vous pouvez vous rendre à l'événement envoi ou ln réception avec la touche F 11. Pour un sémaphore, en appuyant sur F11 sur un "SemaphoreTake" appel prendriez-vous pour l'SemaphoreGive appel correspondant. Pour envoyer une file d'attente, F11 vous amène à l'événement où les données particulières sont reçues.

Pour bloquer événements d'appel du noyau, en utilisant F12 vous pouvez montrer la raison du déclenchement / libéré de blocage, si cela est dû à un appel système. C'est également disponible dans l'affichage d'information Acteur ("déclenchée par"). Cela donne dans de nombreux cas le même résultat que lors de l'utilisation F11, mais pas toujours. Par exemple, si une file d'attente envoyée blocs en raison d'une file d'attente pleine, F12 affiche la plus récente réception de l'évènement depuis que permet l'opération d'envoi, alors que F11 montre l'endroit où le message reçu a été envoyé.

## IV.3.2. Les options de menu

### IV.3.2.1. Fichier

Le menu Fichier comporte des options standard comme trace ouverte, rechargez trace actuelle et a récemment ouvert des fichiers de trace. Il comprend également Export vue en cours sur l'image, données Acteur d'exportation et outils externes.

Exporter la vue en cours image vous permet d'exporter la vue de trace actuelle à une image, par exemple, de la documentation ou pour partager une question avec des collègues.

Données Acteur d'exportation vous permet d'exporter les données d'instance de l'acteur dans un fichier texte. Les données exportées contient l'heure de début, heure de l'exécution, le temps de réponse et la fragmentation de chaque instance des acteurs sélectionnés. Vous pouvez l'exporter en tant que champs séparés par des tabulations pour l'importation dans d'autres outils ou de l'espace des colonnes alignées pour une lecture facile.

Outils externes vous permettent de définir des outils qui peuvent être lancés à partir de l'intérieur FreeRTOS + Trace. Cela peut par exemple être utilisé pour lancer un outil ou un script qui télécharge les traces de votre dispositif, à savoir en faisant votre débogueur prenant un dépotoir de RAM.

### IV.3.2.2. Trouver

Le menu Rechercher vous permet de trouver rapidement l'instance précédente ou suivante à partir de l'instance sélectionnée. Il permet également d'ouvrir la boîte de dialogue du Finder (peut également être ouvert en utilisant CTRL-F).

### IV.3.2.3. Voir

Le menu Affichage vous permet de configurer l'affichage de trace et d'ouvrir d'autres points de vue. Voir la page respective pour des informations sur les autres points de vue.

Nouvelle fenêtre de Trace ouvre une nouvelle vue de trace.

Détails de trace montre propriétés enregistreur connexes de la trace actuelle, tels que le nombre d'événements et la longueur de la trace, ainsi que des détails techniques supplémentaires.

Trace Paramètres d'affichage vous permet de configurer l'affichage de trace, comme les couleurs d'acteurs et d'afficher ou non les signets ou non.

Trace Mode d'affichage vous permet de définir le mode d'affichage, tel que présenté dans la vue Modes sujet ci-dessus.

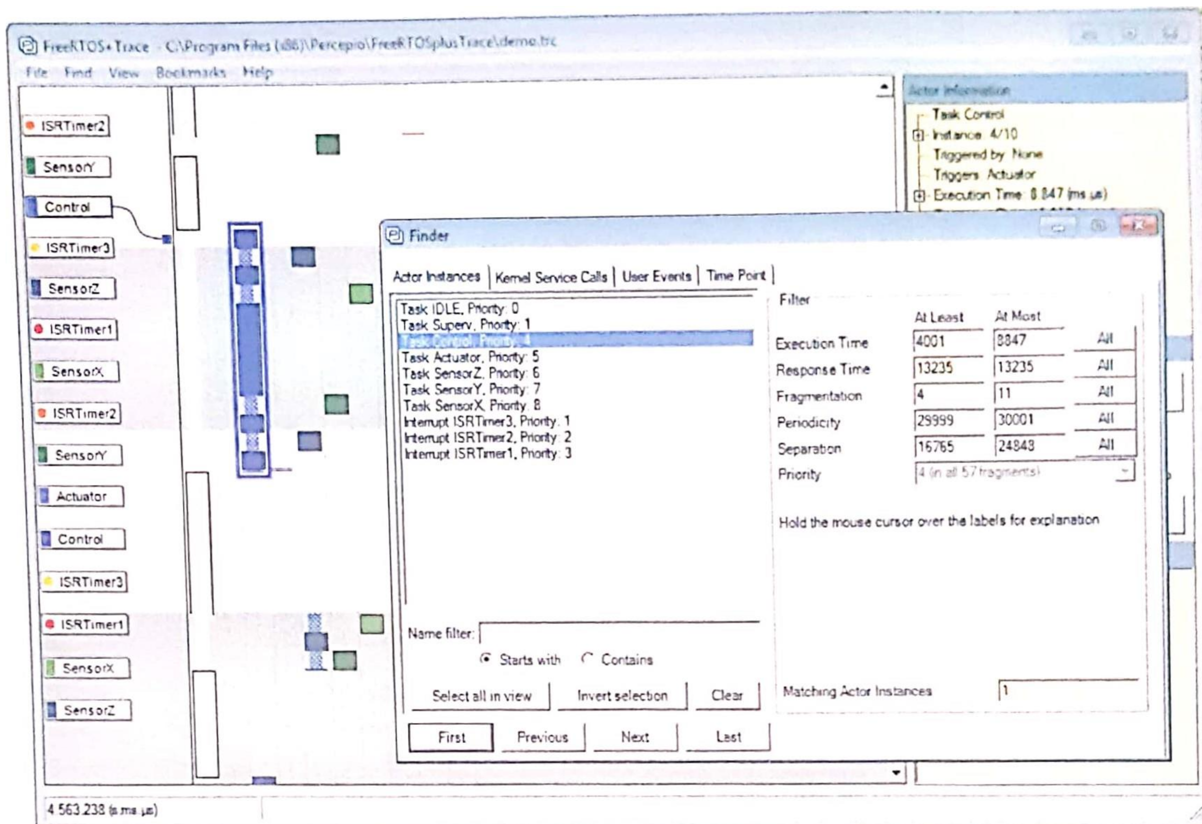
Unité de temps vous permet de changer l'unité de temps utilisée dans l'application.

Time Mode vous permet de changer le point zéro relatif dans le temps.

Afficher le panneau d'outils vous permet de basculer la visibilité de la palette d'outils sur la droite. Cela peut être utile si vous souhaitez que la fenêtre d'application plus petit, par exemple, en ayant de multiples points de vue ouvert. Le panel d'outil peut être restauré à l'aide de la vue menu contextuel de la trace.

## IV.4. Fenêtre du Finder

La fenêtre Finder permet de trouver rapidement les instances de l'acteur, appels de service du noyau et des événements de l'utilisateur, à l'aide de divers filtres. Le Finder permet également d'accéder à un point donné dans le temps.



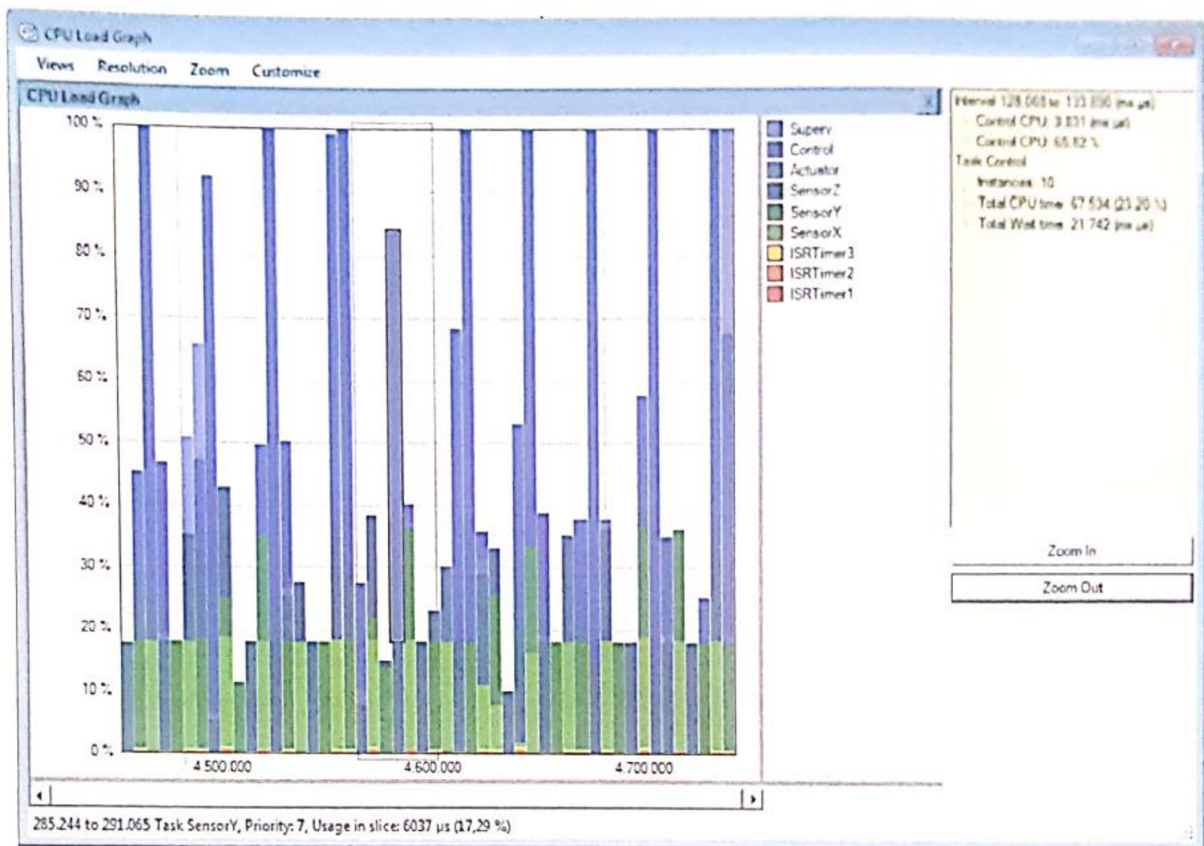
## IV.5. Horizontale Trace View

Cette vue montre une visualisation horizontale de Gantt mode de la trace d'exécution acteur. Cela permet de corréler la trace acteur avec d'autres vues horizontales, telles que le graphique de charge CPU, l'intensité de planification ou de l'utilisation des objets du noyau.

On peut combiner ce point de vue avec d'autres vues horizontales en choisissant Vues -> Ajouter -> (type de graphique).

## IV.6. Charge CPU Graphique

La charge du processeur graphique affiche l'utilisation du CPU au fil du temps, par acteur ou global. Par défaut, il affiche tous les acteurs sauf la tâche d'inactivité(ldle). L'analyse fonctionne en divisant la courbe en un certain nombre d'intervalles, la valeur par défaut est de 100 intervalles. L'utilisation du processeur pour un acteur dans un intervalle est la quantité de temps CPU utilisé par l'acteur dans cet intervalle, divisée par la longueur de l'intervalle. Pour chaque intervalle, tous les acteurs d'exécution en ce que l'intervalle seront tirées comme des rectangles empilés les uns sur les autres. La hauteur du rectangle de chaque acteur représente l'utilisation du processeur pour que l'acteur dans cet intervalle de temps et la hauteur combinée est l'utilisation totale du CPU pour cet intervalle.



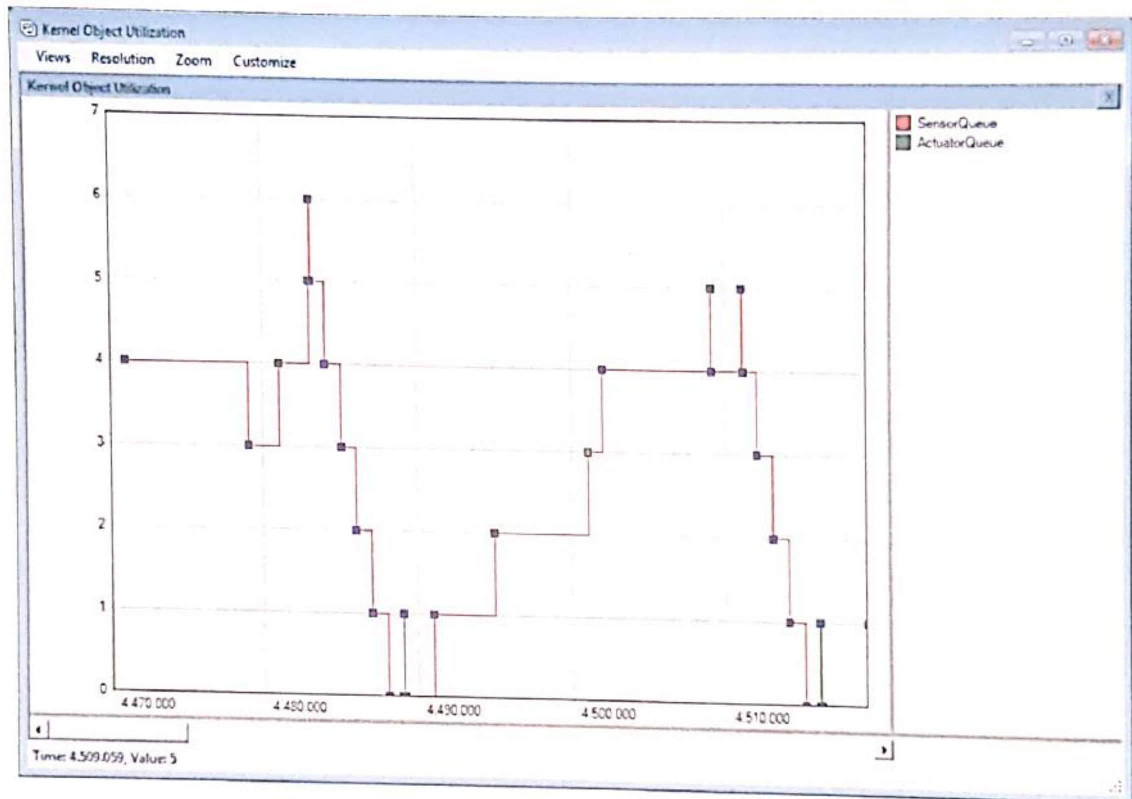

Pour personnaliser l'affichage, utilisez le menu Select Acteur (s). Là, vous pouvez sélectionner les acteurs que vous souhaitez inclure dans la vue. Si le graphe est «bruyant» (due à de courts intervalles), vous pouvez réduire le nombre d'intervalles dans le menu de résolution. Si vous désirez un graphique plus détaillé, vous pouvez augmenter le nombre d'intervalles en sélectionnant une résolution plus élevée. Si vous souhaitez mettre l'accent sur une zone spécifique, vous pouvez cliquer et faire glisser pour sélectionner un intervalle et utiliser le menu Zoom ou cliquez droit pour zoomer sur elle. Vous pouvez également utiliser le zoom ou le menu clic droit pour afficher la sélection ou de la vue actuelle dans la vue de trace ou dans toutes les vues ouvertes.

En cliquant sur un acteur dans le graphique affiche des informations sur cet acteur dans l'intervalle cliqué et en double cliquant un acteur centre la vue sur la trace sur cet intervalle. Un rectangle encadré gris indique l'intervalle de la vue de trace actuelle. Ce rectangle peut être très étroite et apparaissent comme une ligne si la vue de trace est faible.

Vous pouvez combiner ce point de vue avec d'autres vues horizontales en choisissant Vues -> Ajouter -> (type de graphique).

## IV.7. Kernel utilisation d'objets

Ce point de vue permet de visualiser l'état des objets du noyau au fil du temps. Il peut être utilisé pour identifier les situations où le système a du mal à traiter toutes les entrées dans un délai raisonnable.




## IV.8. Intensité de planification

Cette vue affiche la quantité de changements de contexte au fil du temps. Par défaut, il montre toute trace divisé en 100 intervalles. Pour chaque intervalle de temps, une barre est tirée à chaque acteur début ou la reprise de l'exécution au moins une fois dans cet intervalle. La hauteur des barres représente le nombre de fois que l'acteur a commencé ou repris l'exécution dans l'intervalle donné, c'est à dire, le nombre de fragments de l'acteur.

Le nombre d'intervalles peut être contrôlé par le menu Résolution. Vous pouvez cliquer et faire glisser pour sélectionner un intervalle et utiliser le zoom ou le menu clic droit pour zoomer sur cette sélection, pour l'afficher en vue de trace ou de le montrer dans toutes les vues ouvertes.

Remarque ters d'un zoom dans la vue de l'intensité de planification, la longueur de l'intervalle diminue de montrer un graphique plus détaillé. Lors d'un zoom dans un lot, ou si votre trace est 22 courte, vous pouvez donc bénéficier d'un espace vide dans le graphique, correspondant à des intervalles où aucun changement de contexte ne se produit.

Vous pouvez combiner ce point de vue avec d'autres vues horizontales en choisissant Vues > Ajouter -> (type de graphique).

## IV.9. Service de noyau Intensité d'appel

Le Service graphe d'intensité d'appel Kernel affiche le numéro de service du noyau appelle au fil du temps. Cette vue vous permet de trouver des points chauds, où il ya beaucoup de service du noyau appels effectués. Les services du noyau individuels sont codés par couleur en utilisant une échelle de couleur distincte, dans la légende de droite. Vous pouvez activer ou désactiver la visibilité des services du noyau dans cette vue en cliquant sur les étiquettes dans la légende.

Vous pouvez combiner ce point de vue avec d'autres vues horizontales en choisissant Vues > Ajouter (type de graphique).

## IV.10. Temps Bloquer noyau Appel

Le blocage d'appels Temps graphique du noyau affiche le noyau blocage fois des appels de service, c'est à dire, le temps entre l'entrée et le retour des appels bloquants. Chaque point de données représente un appel de service spécifique au noyau, où la position x indique le point dans le temps et la position y le temps de blocage. En cliquant sur un point de données

(un appel de service) mettra en évidence dans la vue de trace. Cela peut être utilisé pour identifier le blocage involontaire, par exemple sur un Mutex, qui pourrait être une cause derrière les temps de réponse inhabituellement élevés de tâches.

**Remarque**
Si accessible à partir du menu Affichage de la fenêtre principale, cette vue tracer tous les appels sur n'importe quel objet du noyau. Vous pouvez faire apparaître le point de vue d'un seul objet par un clic droit sur un appel de service du noyau dans la vue de trace et en sélectionnant Blocage Time Graph de (Nom de l'objet) sous-menu.

Vous pouvez combiner ce point de vue avec d'autres vues horizontales en choisissant Vues > Ajouter -> (type de graphique).

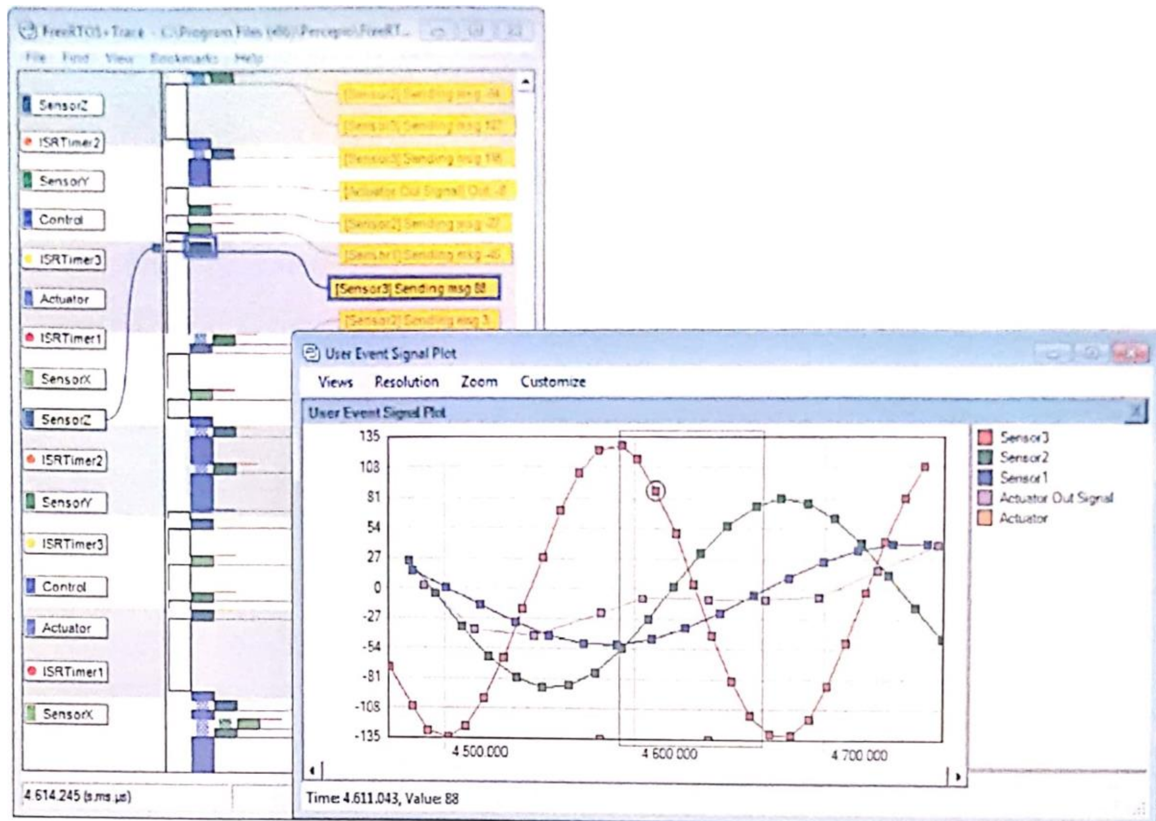
## IV.11. Utilisation Signal Plot événement

Le point de vue de la parcelle de signal affiche une parcelle de données enregistrées à l'aide d'événements utilisateur. Ceci peut être utilisé pour l'analyse des signaux de capteur et le comportement du système de commande. Comme d'autres points de vue, cela est lié à la vue de la trace principale, ce qui permet de corréler des valeurs de signal à l'ordonnancement de tâches et d'autres événements enregistrés. Par exemple, si vous constatez une anomalie dans le signal d'une boucle de régulation de sortie surveillée, vous pouvez cliquer sur le point de données pour montrer le cas spécifique de l'utilisateur dans la vue de trace. Cela vous permet d'analyser ce qui se passait à ce moment - peut-être la programmation a causé un retard supplémentaire de la tâche de boucle de régulation qui a affecté le signal.

Événements de l'utilisateur peuvent être créés en utilisant vTraceUserEvent et en utilisantvTracePrintF . Ce dernier permet des impressions formatées, comme un printf classique:

```
  / * Code d'initialisation - créer l'étiquette de canal pour un
VTracePrintF * /
  traceLabel adc_user_event_channel1 = xTraceOpenLabel («ADC 1");

  / * Un événement utilisateur avancé, à l'emplacement du code
correspondant * /
  vTracePrint (adc_user_event_channel1, "ADC 1:% volts LF", adc_volts);
```




**Remarque**

Si plusieurs arguments de données sont enregistrées dans un seul appel vTracePrintF, seul le premier argument est utilisé dans le complot. Pour tracer plusieurs signaux, utiliser des appels vTracePrintF distincts sur différents canaux d'événement utilisateur.

Vous pouvez combiner ce point de vue avec d ' autres vues horizontales en choisissant Vues > Ajouter -> (type de graphique).

## IV.12. Acteur instance Graphiques

La fenêtre graphique Acteur instance affiche un graphique montrant les propriétés de synchronisation des instances acteur: temps d'exécution, temps de réponse, l'interférence de réponse et la fragmentation. Chaque point de données dans le graphique représente une instance acteur spécifique, où l'axe des abscisses représente l'heure de début d'instance et l'axe des ordonnées la valeur de la propriété.

En cliquant une instance tracé montre et le mettre en surbrillance dans la vue de trace. Vous pouvez filtrer les acteurs en utilisant le menu Filtre ou en cliquant sur un acteur de la légende à droite.

Si vous souhaitez vous concentrer sur une zone spécifique, vous pouvez cliquer et faire glisser pour sélectionner un intervalle et utilisez le menu Zoom ou clic droit pour zoomer dessus. Vous pouvez également utiliser le zoom ou faites un clic droit menu pour afficher la sélection ou la vue actuelle dans la vue de trace ou dans toutes les vues ouvertes.

Ce point de vue peut être utilisée pour identifier les tâches de longue durée à l'aide des graphes de temps d'exécution et de réaction, ou des problèmes potentiels de planification en utilisant la fragmentation et graphiques interférences.

Les propriétés qui peuvent être tracées sont les suivantes:

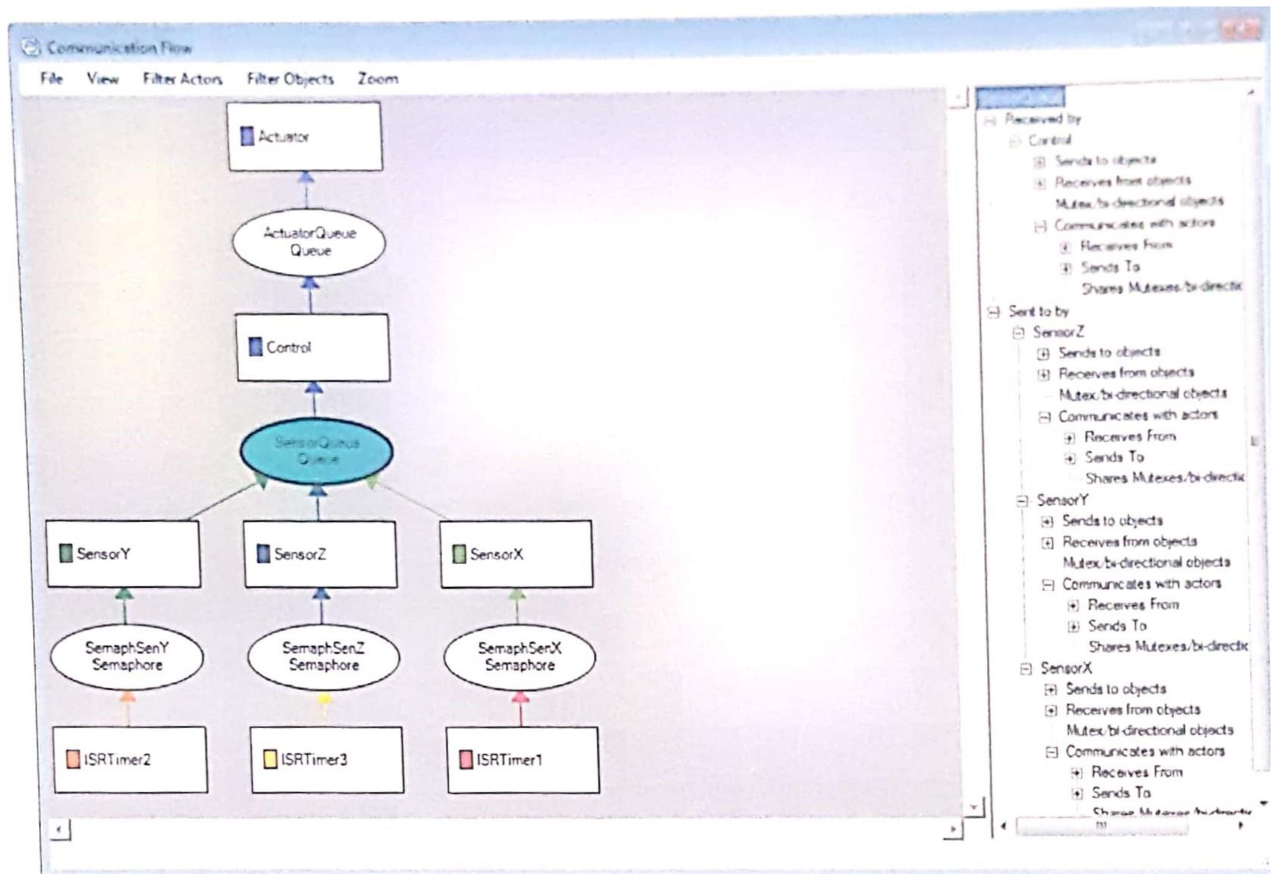
- Temps d'exécution: La quantité de temps processeur utilisé par une instance Acteur, à l'exclusion préemptions.
- Temps de réponse: Le temps entre le début d'une instance acteur jusqu'à ce qu'elle se termine.Le temps de réponse pour les tâches est compté à partir du moment où la tâche devient prêt à exécuter (par exemple, le point où le noyau définit l'état de programmation de la tâche à READY). Pour les routines de service d'interruption, le temps de réponse est comptée à partir du début de l'exécution. Notez que les produits Tracealyzer avec des enregistreurs Percepio-développés (par exemple FreeRTOS + Trace) peut permettre d'exclure les événements prêts à réduire l'utilisation de mémoire RAM. Dans de tels cas, temps de réponse des tâches est compté à partir du début de l'exécution.

- Temps d'attente: Il s'agit Temps de réponse - Temps d'exécution, c'est à dire, le temps dans une instance où l'acteur ne s'exécute pas.
- Temps de démarrage: C'est le temps entre le début prêts et d'exécution.
- Interférences Réponse: La relation entre le temps d'exécution et temps de réponse. Une valeur de 30% signifie que le temps de réponse est de 30% plus longtemps que le temps d'exécution, c't est à dire, en raison de tâches anticipation des interruptions, ou blocages. Une valeur de 0% signifie que le temps de réponse est égal au temps d'exécution, à savoir que l'instance acteur complètement exécutée sans changement de contexte.
- Fragmentation: Le nombre de fragments dans une instance Acteur, soit, en raison de changements de contexte. Si une instance Acteur exécute en totalité sans préemptions, de la fragmentation de l'instance est 1.
- Séparation: Le temps entre l'heure de début et l'instance finition exemple précédent. Disponible en deux versions, selon l'endroit où de compter le temps de démarrage par exemple: du Prêt etdu début d'exécution .
- Périodicité: le temps entre l'heure de début d'instance et le temps de démarrage instance précédente. Disponible en deux versions, selon l'endroit où de compter le temps de démarrage par exemple: du Prêt et du début d'exécution .

Vous pouvez combiner ce point de vue avec d'autres vues horizontales en choisissant Vues > Ajouter -> (type de graphique).

## IV.13. Débit de communication

Le graphique des flux de communication offre un aperçu rapide de la communication et de synchronisation entre les acteurs d'une trace, à travers les files de messages, les sémaphores et les mutex. Ce graphe peut être généré sur l'ensemble de la trace, ou pendant un intervalle spécifique uniquement. Par exemple, vous pouvez utiliser le CPU charge spectateur de graphiques pour identifier un sommet où il ya beaucoup d'activité, puis sélectionnez l'intervalle à l'aide d'une sélection de glisser de la souris, faites un clic droit et sélectionnez "Afficher le flux de communication" dans le menu contextuel pour afficher la communication diagramme d'écoulement de cet intervalle spécifique.




Les acteurs sont affichés sous forme de rectangles et d'autres objets du noyau sont présentés comme des ellipses ou des hexagones. Ellipses sont utilisés pour les files d'attente de messages et d'autres opérations de passage de messages, à savoir, la communication, tout en hexagones sont des mutex et serrures similaires, par exemple pour des objets de synchronisation.
Il ya deux modes dans ce point de vue:

- Acteurs et objets: affiche tous les objets et les dépendances, à la fois acteur et de communication dépendances sur les objets de synchronisation, comme les mutex.
- Seulement Acteurs: Dans ce mode, seuls les acteurs ont montré que les dépendances de communication à d'autres tâches. Dépendances sur les objets de synchronisation (mutex par exemple) ne sont pas représentés.

La classification des objets du noyau dans des objets de communication (des ellipses) et des objets de synchronisation (hexagones) est basée sur le service de noyau effective appels trouvée, et non le type statique de l'objet. Cette depuis sémaphores peut être utilisé à des fins, comme un signal directionnel (une forme de communication), et comme un verrou de synchronisation, c'est à dire, comme un Mutex. Un objet du noyau est traitée comme un objet de synchronisation (hexagone) si il ya des acteurs qui à la fois augmenter et diminuer l'état d'objet (par exemple, faire à la fois un "LockMutex" et "ReleaseMutex'l). Cela signifie qu'un sémaphore peut apparaître comme un hexagone (si l'un des acteurs les deux signaux et attend pour lui) tandis qu'un mutex dans de rares cas pourrait apparaître comme une ellipse, si un seul type d'opération ("LockMutex" ou "ReleaseMutex") ont sont enregistrées sur l'objet particulier.

### IV.13.1. Appliquer des filtres

Parfois, il est utile pour filtrer les tâches ou des objets spécifiques, tels que si vous avez un débogage ou d'une tâche de rédacteur du journal que de nombreuses autres tâches envoient des données. Par défaut, tous les acteurs et les objets qui n'effectue aucune communication est masqué, mais vous pouvez ajuster ces réglages en utilisant les menus de filtrage. Vous pouvez également cliquer sur un noeud de le cacher.

Quand un clic droit sur un nœud, vous aurez également présentés avec des options pour afficher uniquement les noeuds connectés, en une, deux ou illimitée étapes. Cela montre que les acteurs et les objets qui affectent ou sont affectées par l'acteur ou l'objet sélectionné via la communication dirigée (communication non dirigé tel que par mutex ne sont pas respectées, même si les objets seront affichés).

Le spectacle tous connectés et le spectacle Tous Seules les options connecté, indiquent les mêmes nœuds (acteurs / objets), mais risque d'afficher des ensembles d'arêtes (lignes). La différence est que le premier ne cache pas la communication entre les nœuds, même si c'est sans rapport avec le nœud sélectionné. Par exemple, si vous avez les chaînes de communication Taska -> QueueB -> taskB -> outputQueue et Taska -> outputQueue et vous montrer les noeuds connectés à taskB , Voir tous connectés seulement (2 étapes) ne devraient seulement la première chaîne tout Afficher tous connectés (2 étapes) pourrait également montrer la deuxième chaîne.

## IV.14. Rapport de Statistique

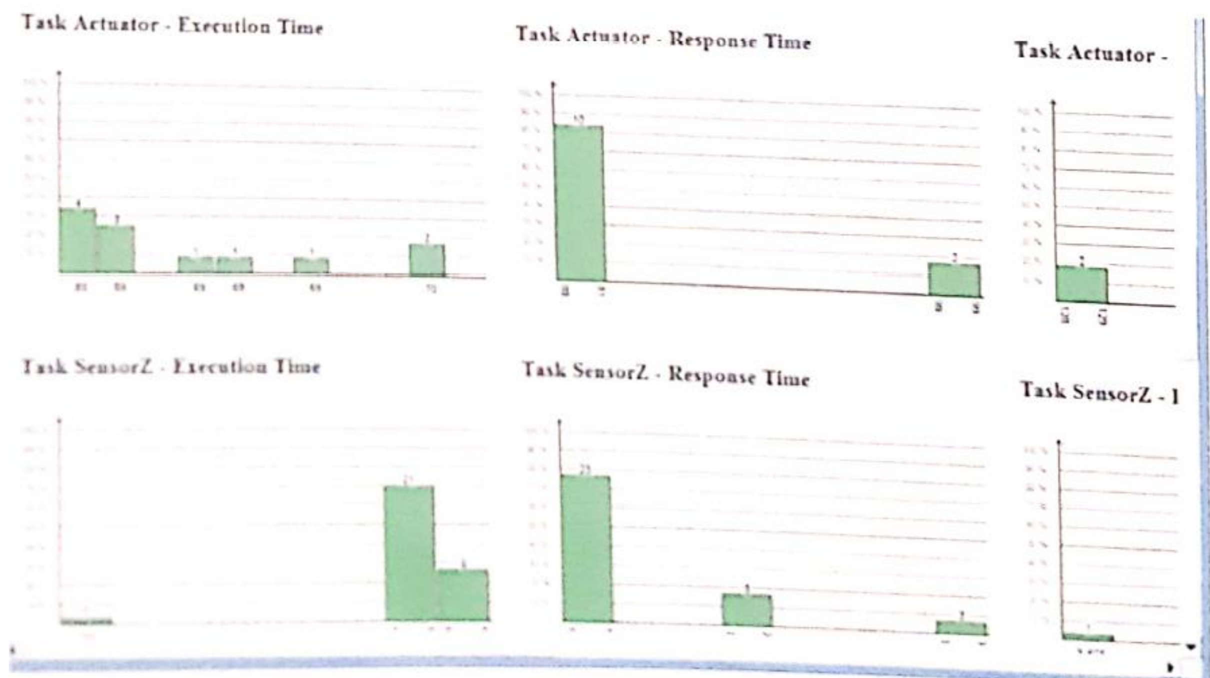
Le générateur de rapports de statistiques vous permet de générer un rapport de haut niveau sur les propriétés de synchronisation et l'utilisation du processeur pendant toute la trace (si n'existe aucune sélection dans la vue de trace principal), ou I [t] intervalle sélectionné. Le rapport peut être enregistré au format HTML, ce qui vous permet de stocker pour référence future, par exemple, de comparer le calendrier et la charge du processeur entre les différentes versions de votre système.

Lorsque vous lancez le rapport statistique, une boîte de dialogue de configuration apparaît où vous pouvez sélectionner les acteurs que vous souhaitez inclure dans le rapport, et quelles sont les propriétés que vous souhaitez inclure. Lorsque vous avez terminé, cliquez sur le bouton Afficher le rapport pour consulter le rapport. Le rapport comporte deux sections, une table présentant un résumé, et (éventuellement) un ensemble d'histogrammes montrant les distributions détaillées. Les histogrammes ne sont inclus que si la case à cocher Afficher les diagrammes graphiques est activé.

| Actor | Priority | | Count | CPU Usage | Execution Time | | | Response Time | | | Periodicity | | | Separation | | | Fra[g] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Min | Max | | % | Min | Avg | Max | Min | Avg | Max | Min | Avg | Max | Min | Avg | Max | Min |
| Task IDLE | 0 | 0 | 1 | 51,4 | - | - | - | - | - | - | - | - | - | - | - | - | 92 |
| Task Superv | 1 | 1 | 2 | 1,34 | 1.882 | 1.949 | 2.017 | 2.047 | 4.062 | 6.076 | 250.000 | 250.000 | 250.000 | 247.953 | 247.953 | 247.953 | 1 |
| Task Control | 4 | 4 | 10 | 23,2 | 4.001 | 6.753 | 8.847 | 5.152 | 8.969 | 13.235 | 29.999 | 30.000 | 30.001 | 16.765 | 21.052 | 24.848 | 4 |
| Task Actuator | 5 | 5 | 12 | 0,284 | 68 | 69 | 70 | 80 | 83 | 99 | 128 | 24.454 | 34.955 | 29 | 24.371 | 34.875 | 1 |
| Task SensorZ | 6 | 6 | 30 | 10,2 | 10 | 992 | 1.034 | 1.034 | 1.362 | 3.123 | 9.974 | 9.999 | 10.001 | 6.877 | 8.626 | 8.942 | 1 |
| Task SensorY | 7 | 7 | 21 | 7,4 | 1.000 | 1.025 | 1.041 | 1.029 | 1.148 | 2.060 | 13.972 | 14.000 | 14.028 | 11.941 | 12.846 | 12.968 | 1 |
| Task SensorX | 8 | 8 | 16 | 5,64 | 1.006 | 1.026 | 1.036 | 1.030 | 1.050 | 1.061 | 17.972 | 18.000 | 18.028 | 16.940 | 16.950 | 16.970 | 1 |
| Interrupt ISRTimer3 | 1 | 1 | 29 | 0,213 | 21 | 21 | 22 | 21 | 21 | 22 | 10.000 | 10.000 | 10.001 | 9.978 | 9.978 | 9.979 | 1 |
| Interrupt ISRTimer2 | 2 | 2 | 21 | 0,154 | 21 | 21 | 22 | 21 | 21 | 22 | 13.972 | 14.000 | 14.028 | 13.950 | 13.978 | 14.007 | 1 |
| Interrupt ISRTimer1 | 3 | 3 | 16 | 0,118 | 21 | 21 | 22 | 21 | 21 | 22 | 17.973 | 18.000 | 18.027 | 17.951 | 17.978 | 18.006 | 1 |

L'histogramme vous montre un ou plusieurs bars, où chaque barre show:

- L'intervalle représenté par la barre (les valeurs de l'axe X)
- Le pourcentage de cas de l'acteur dans cet intervalle (les valeurs de l'axe Y)
- Le nombre absolu d'occurrences à l'intérieur de cet intervalle (la valeur à virgule flottante au-dessus de la barre.



Notez qu'une barre peut représenter une instance unique acteur dans certains cas. Dans ce cas, seule la valeur unique est représentée sur l'axe X, au lieu des limites d'intervalle. A notor également que deux barres adjacentes peuvent cacher "espace vide" dans la distribution de densité sous-jacente, c'est à dire qu'il peut y avoir un intervalle avec aucun cas de l'acteur correspondance entre eux.

L'histogramme bars et la plupart des champs de la table sont liés à la boîte de dialogue du Finder, par exemple les zones indiquées en rouge sur l'illustration ci-dessus. La valeur de la table et la barre de l'histogramme fois représente le temps de réponse le plus élevé constaté une certaine tâche. Lorsque l'un d'entre eux sont cliqués, le dialogue est ouvert Finder et le filtre d'acteur est réglée pour trouver ce cas précis. Le Finder va alors montrer ce match dans la fenêtre de trace principale.

## IV.15. Journal des événements utilisateur

La vue User Log d'événements affiche tous les événements utilisateur de l'enregistrement dans une liste, ce qui correspond à une console virtuelle . Ce point de vue peut être ouvert en double-cliquant sur un événement utilisateur dans la vue trace principale, ou dans le menu Affichage. Les événements utilisateur peuvent être recherchés et filtrés en utilisant la zone de texte dans le panneau sur la droite.

## IV.16. Historique de l'objet du noyau

Le point de vue de l'histoire de l'objet du noyau affiche tous les appels de service du noyau pour un objet spécifique du noyau. Ce affiché sous forme de liste, contenant l'acteur de faire l'appel, le type d'événement et de l'état de l'objet du noyau.

Pour les objets de file d'attente, la longueur actuelle de la file d'attente est affiché dans la colonne de droite, avec des numéros de séquence sur les messages individuels (les rectangles). Les couleurs des messages indiquent l'acteur qui a envoyé le message. Si vous sélectionnez un succès envoyer ou recevoir des cas, il est possible de suivre le message au destinataire ou l'expéditeur en utilisant les boutons dans le panneau de l'outil sur la droite, "Aller envoi de l'événement" et "Aller à recevoir l'événement." Cela vous permet de suivre les messages envoyés entre les instances de tâches individuelles, par exemple, de trouver et d'étudier l'endroit où un message de la file d'attente avec des données incorrectes a été envoyé.

La liste des événements peut être filtrée pour afficher uniquement les événements d'un certain service ou à partir d'une certaine tâche, en utilisant les deux menus Filtre. En cliquant sur un événement affiche les détails à ce sujet dans le panneau de droite, qui dispose également de boutons de navigation. Double-cliquer sur un événement mettra en évidence dans la vue de trace.

# *V Chapitre 5 : APPLICATION*

# V.1. Application FreeRTOS + Trace et FreeRTOS + CLI

## V.1.1. Utilisation de la FreeRTOS Win32 Simulator

### V.1.1.1. Présentation

Cette section décrit une application qui fonctionne dans le simulateur Win32 FreeRTOS. En utilisant le simulateur, il est facile d'évaluer FreeRTOS + CLI et FreeRTOS + Trace sur un PC de bureau standard, en utilisant les outils de développement gratuits, et sans avoir besoin de connecter un équipement externe.

Pour conserver tout aussi simple que possible, les FreeRTOS + interface de ligne de commande CLI est accessible via un socket UDP sur le bouclage adresse IP par défaut de Windows de 127.0.0.1. Utilisation de l'adaptateur de bouclage ce qui permet à l'application de fonctionner sur un seul ordinateur, et sans connexion réseau.

Les commandes suivantes sont mises en œuvre :

| **Chaîne de commandement** | **Comportement de commande** |
|---|---|
| Trace | La commande trace est utilisée pour démarrer et arrêter l'enregistrement de trace. FreeRTOS + CLI<br><br>Pour démarrer l’enregistrement de trace :<br><br>"Trace start"<br><br>Pour arrêter l’enregistrement de trace :<br><br>"Tracer stop"<br><br>Sauvegardé dans une mémoire tampon |
| Echo_parameters | "Echo_parameters un deux trois quatre"<br><br>Ensuite, la sortie générée sera :<br><br>Les paramètres sont les suivants :<br>1 : un<br>2 : deux<br>3 : trois<br>4 : quatre |

| | |
|---|---|
| Echo_3_parameters | Une commande qui prend un nombre exact de paramètres. Lorsque la commande est acceptée, la mise en œuvre de la commande écho à tous les trois paramètres d'une manière similaire à la commande echo_parameters détaillé ci-dessus. |
| Task-stats | Affiche un tableau, chaque ligne qui affiche des informations d'état sur une tâche unique. |
| Run-time-stats | Affiche un tableau, chaque ligne indique la quantité de temps par tâche dans l'état marche. Autrement dit, le temps d'exécution alloué à chaque tâche. |

### V.1.1.2. Fonctionnalité

Deux tâches et une seule file d'attente sont utilisées pour générer un modèle d'exécution simple, qui peut être consulté dans les FreeRTOS + Trace interface graphique.

Une file d'attente de priorité faible envoyer tâche envoie à plusieurs reprises un message à la file d'attente. Une file d'attente de priorité plus élevé reçoivent tâche essaie à plusieurs reprises de lire un message de la file d'attente, le blocage sur la file d'attente opération de lecture lorsque aucun message n'est disponible. Une explication du modèle d'exécution qui en résulte est fournie ci-dessous.

Une tâche de surveillance de la trace est également créée qui affiche un message lorsque celui-ci détermine que l'état de l'enregistreur trace n'a changé depuis la dernière fois exécutée.

Il convient de noter que, parce que le simulateur Windows est utilisé, les informations de synchronisation s'affiche lorsque l'application est en cours d'exécution, et enregistré dans le journal de suivi, n'ont pas unités significatives.

## V.1.2. Construire et exécuter l'application

### Section main ()

```
/* Standard includes. */
include <stdio.h>

#include <stdint.h>

/* FreeRTOS includes.*/
#include <FreeRTOS.h>

include "task.h"

include "queue.h"

/* FreeRTOS+Trace includes. */
include "trcUser.h"

/* Priorities at which the tasks are created. */
#define nainQUEUE_RECEIVE_TASK_PRIORITY          ( tsklDLE_PRIORITY + 2 )
#define mainQUEUE_SEND_TASK_PRIORITY        ( tsklDLE_PRIORITY + 1 )

#define mainUDP_CLI_TASK_PRIORITY                ( tsklDLE_PRIORITY )
/* The rate at which data is sent to the queue. The (simulated) 50ms value is
converted to ticks using the portTlCK RATE_MS constant. */

#define nainQUEUE_SEND_FREQUENCY_MS              ( 50 / portTICK_RATE_MS )

  /* The number of items the queue can hold. This is 1 as the receive task
will remove items as they are added, meaning the send task should always find
the queue empty. */

#define mainQUEUE_LENGTH
/*……………………………………….*/

/*



 *The queue send and receive tasks as described in the comments at the top of
 *this file.



static void prvQueueReceiveTask( void *pvParameters );
```

```
static void prvQueueSendTask( void *pvParameters );

/*
*The task that implements the UDP command interpreter using FreeRTOS+CLl.
*/
extern void vUDPCommandlnterpreterTask( void *pvParameters );

/*

*Register commands that can be used with FreeRTOS+CLl through the UDP socket.



*The commands are defined in CLI-commands.c.
*/

extern void vRegisterCLlCommands( void );

/*The queue used by both tasks.*/

static xQueueHandle xQueue = NULL;
/*..........................................*/
int main( void )

{
const uint32_t ulLongTime_ms = 250UL;


        /* Create the queue used to pass messages from the queue send task to the
        queue receive task.*/

        xQueue = xQueueCreate( mainQIJEUE LENGTH, sizeof( unsigned long ) );
        /* Give the queue a name for the FreeRTOS+Trace log.*/
        vTraceSetQueueName( xQueue, "DemoQ" );

        /* Start the two tasks as described in the comments at the top of this file.*/
        xTaskCreate( prvQueueReceiveTask,    /* The function that implements the task.*/

        ( signed char * ) "Rx", /* The text name assigned to the task - for debug only as it is not used by the kernel. */

                configMINIMAL_STACK_SIZE, /* The size of the stack to allocate to the task. Not actually used as a stack in
                                                the Win32 simulator port.*/

         NULL,
        /* The parameter passed to the task - not used in this example.*/

   mainQUEUE_RECEIVE_TASK_PRIORITY, /* The priority assigned to the task. */
```

```
NULL);
   /*The task handle is not required, so NULL is passed.*/

xTaskCreate( prvQueueSendTask, ( signed char  * ) "TX", configMINIMAL_STACK_SIZE, NULL,
   mainQUEUE_SEND_PRIORITY,NULL);

   /*Create the task that handles the CLI on a UDP port. The port number is set using the configUDP_CLI_PORT_NUMBER setting
   in FreeRTOSConfig.h. */

xTaskCreate( vUDPCommandlnterpreterTask, ( signed char * ) "CLI", configMlNlMA_STACK_SIZE,
NULL,mainUDP_TASK_PRIORITY,NULL);

/* Create the task that monitors the trace recording status, printing periodic information to the display.*/
vTraceStartStatusMonitor();

/*Register commands with the FreeRTOS+CLl command interpreter.*/
vRegisterCLlCommands();

/*Start the tasks and timer running.*/
vTaskStartScheduler();

   /*  If all is well, the scheduler will now be running, and the following

line will never be reached. If the following line does execute, then

there was insufficient FreeRTOS heap memory available for the idle and/or

timer tasks             to be created. See the memory management section on the

FreeRTOS web site for more details (this is standard text that is not not

really applicable to the Win32 simulator port).*/

  for( ;; )
  {

           Sleep( ulLongTime_ms );

  }

}

/*……………………………………………….*/

 Static void prvQueueSendTask( void *pvParameters )
 {
 portTickType xNextWakeTime;


  Const unsigned long ulValueToSend = 100UL;

           /* Remove warning about unused parameters.*/
```

```
( void ) pvParameters;
   /*Initialise xNextWakeTime - this only needs to be done once.*/

xNextWakeTime = xTaskGetTickCount();

for(;; )
           {

                /*Place this task in the blocked state until it is time to run again.

             While in the Blocked state this task will not consume any CPU time.*/

             vTaskDelayUntil( &xNextWakeTime, mainQUEUE_SEND_FREQUENCY_MS );

             /* Send to the queue - causing the queue receive task to unblock and
             write a message to the display. O is used as the block time so the

             sending operation will not block - it shouldnt need to block as the

            queue should alwaysbe empty at this point in the code, and it is an

             error if it is not.*/

             xQueueSend( xQueue, &ulValueToSend, OU );

             }

}

/*……………………………………………….*/

static void prvQueueReceiveTask( void *pvParameters )
{
unsigned long ulReceivedValue;
       /* Remove warning about unused parameters.*/
     ( void ) pvParameters;

              for( ;; )
               {
                     /* Wait unfil something arrives in the queue - this task will block
                        indefinitely provided INCLUDE_vTaskSuspend is set to 1 in
                        FreeRTOSConfig.h.*/

                   xQueueReceive( xQueue, &ulReceivedValue, portMAX_DELAY );

                        /*To get here something must have been received from the queue, but

                              is it the expected value? If it is, write the message to the
                                display before looping back to block on the queue again.*/
```

```
                if( ulReceivedValue == 100UL )
                {
                    printf( "Message received!\r\n" );

                    ulReceivedValue = OU;
                }


        }


}


/*……………………………………………*/

void vApplicationIdleHook( void )
{
const unsigned long ulMSToSleep = 5;


        /* This function is called on each cycle of the idle task if
        configUSE_IDLE_HOOK is set to 1 in FreeRTOSConfig.h. Sleep to reduce CPU load.*/

        Sleep( ulMSToSleep);


}


/*……………………………………………*/

void vAssertCalled( void )
{


const unsigned long ulLongSleep = 1000UL;

        taskDISABL_INTERRUPTS();

        for( ;; )
        {
           Sleep( ulLongSleep );
        }


}



/*………………………………………………………………*/
```

La console de commande utilise un socket UDP sur l'adresse IP 127.0.0.1 et le port 5001 pour recevoir une entrée ligne de commande, et un socket UDP sur la même adresse IP et le port 5002 pour la sortie. Un programme de console UDP, tel que l'utilité libre YAT, peut être utilisé comme une interface UDP. Notez que 127.0.0.1 est l'adresse IP de bouclage, si une connexion réseau n'est pas nécessaire.

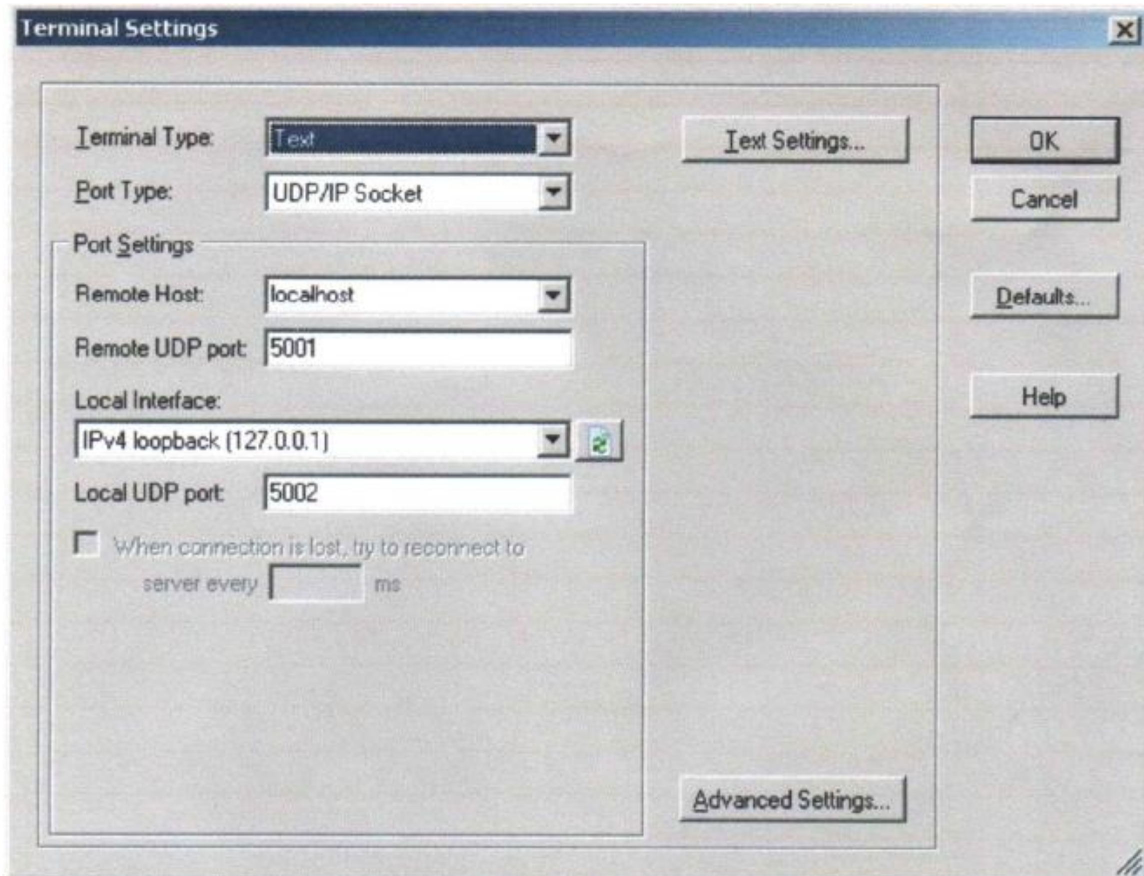



Figure V-1 : Les paramètres du terminal requis YAT

## V.2. Création d'un enregistrement Trace FreeRTOS +

Pour créer un enregistrement de trace :

Testez la connexion entre le terminal UDP UDP et l'application qui s'exécute en essayant de la "task-stats" et "run-time-stats" des commandes.

Démarrer l'enregistreur de trace en entrant le "début de trace" de commande dans le terminal UDP. Laissez l'enregistrement en cours pour environ cinq à dix secondes, puis terminer l'enregistrement en entrant la « fin de trace » de commande.

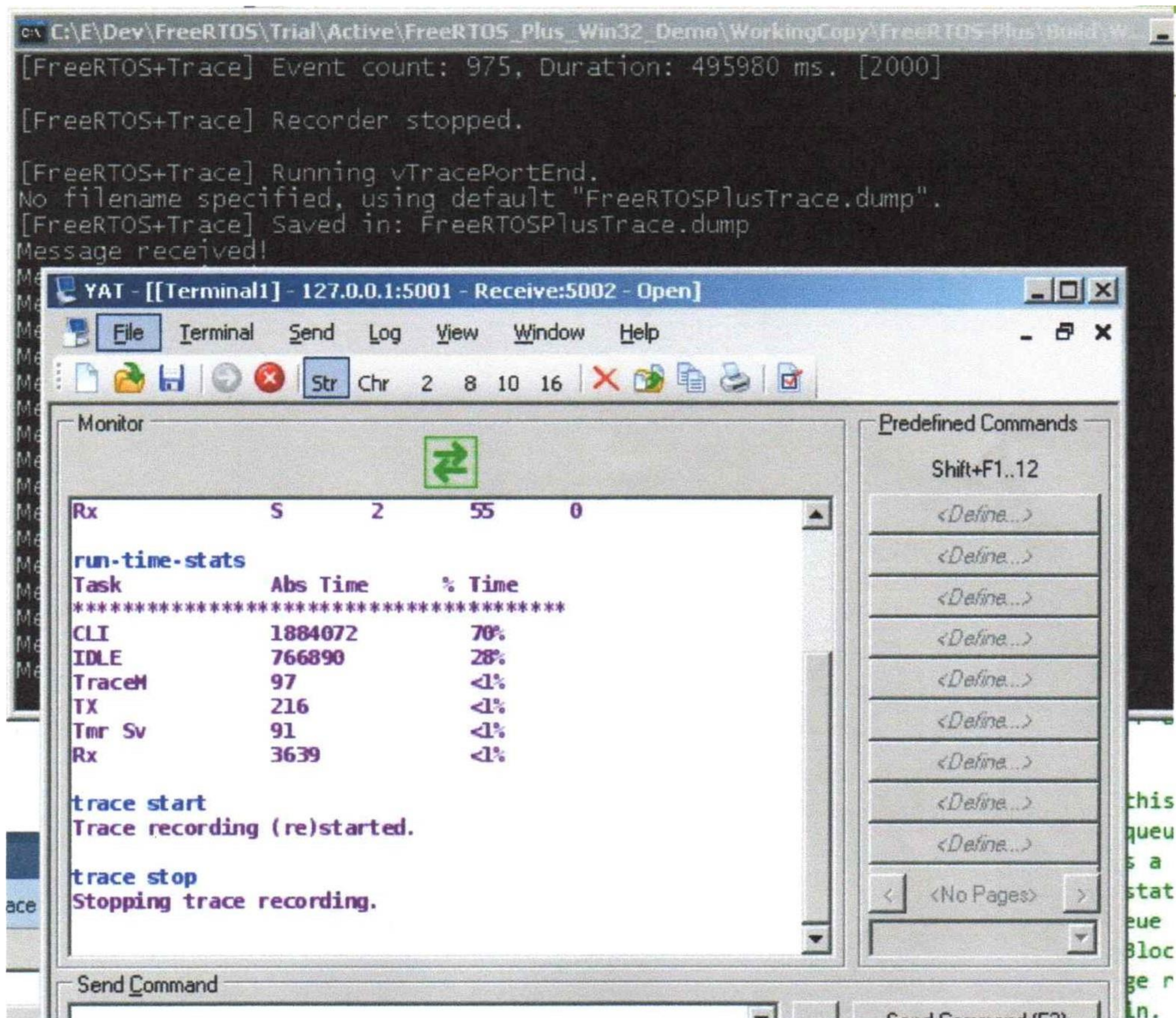



Figure V-2 :Capture d'écran après la trace RTOS a été démarré et arrêté à partir de la console UDP

## V.2.1. Voir l'enregistrement Trace dans FreeRTOS +

Pour ouvrir et visualiser l'enregistrement de trace :

Ouvrez le fichier de trace à l'intérieur des applications FreeRTOS + Trace. Le fichier de trace a été enregistré en tant que FreeRTOSPlusTrace.dump dans le répertoire qui contient le projet. Les données de suivi sont affichées dans les fenêtres principales et FreeRTOS trace. Défiler vers le bas jusqu'à ce que la trace afficher vous remarquez "TX[I]' et "RX" marqueurs sur le côté gauche de l'écran. Ce sont les périodes pendant lesquelles la file d'attente d'émission et de réception file d'attente des tâches ont été respectivement de l'exécution. Zoom sur la région se traduira par un affichage similaire à la capture d'écran ci-dessous. Notez que la capture d'écran a tous les

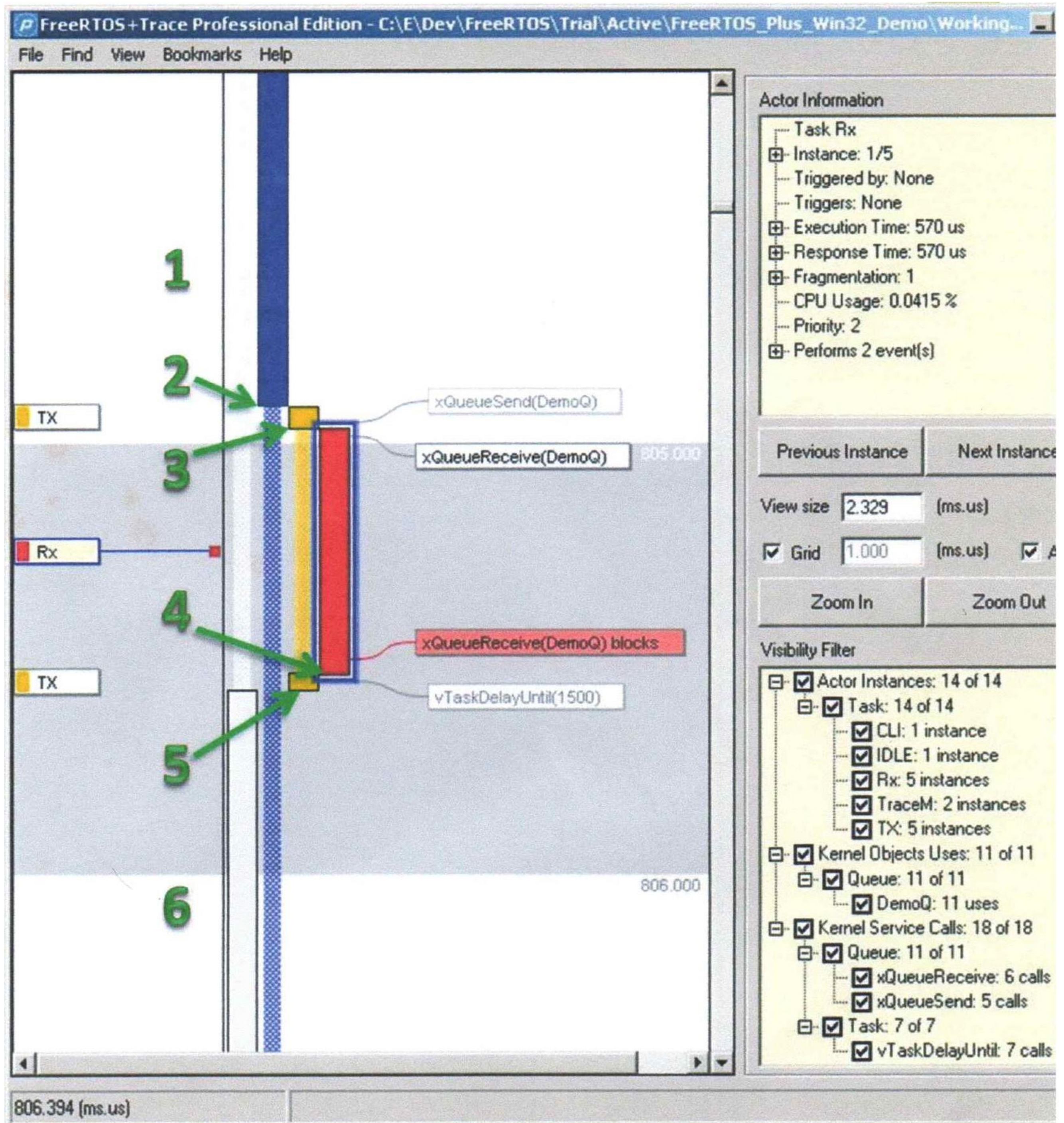



Figure V-3 : Voir le RTOS enregistré trace dans FreeRTOS + Trace avec annotation explicative

L'image ci-dessus a été annoté avec des numéros verts pour mettre en évidence les points d'intérêt. Se référant à l'image ci-dessus :

1. A (1) la tâche CLI est en marche. Dans un scénario réel, sur du matériel réel, la tâche CLI suffit d'exécuter quand il est entré à traiter. Dans cet environnement simulé, la pile TCP / IP ne peut pas être autorisé à bloquer, la tâche CLI est toujours disponible pour l'algorithme.

2. A (2) de la tâche TX (la file d'attente envoyer tâche) débloque parce qu'il est temps pour lui d'envoyer un autre message à la file d'attente. La tâche TX préempte la tâche 12 CLI.
3. A (3) la tâche TX appelle xQueueSend () pour envoyer un message à la file d'attente DemoQ. La plus grande priorité des tâches RX (attente dans la file des tâches) a été bloqué dans la file d'attente d'un message arrive, alors maintenant débloque et préempte la tâche Tx.
4. A (4) la tâche RX appelle xQueueReceive () à nouveau, mais la file d'attente est vide, alors une fois de plus la tâche RX rentre dans l'état bloqué (signifiée par la couleur rouge de la xQueueReceive () étiquette dans la vue trace) permettant le TX tâche d'entrer dans l'état Running nouveau.
5. A (5) de la tâche TX appelle vTaskDelayUntil () pour entrer dans l'état bloqué jusqu'à ce que de nouveau il est temps pour lui d'envoyer un message à la file d'attente.
6. À (6) de la tâche de fond est en cours d'exécution.

# ***<u>Conclusion générale</u>***

Les systèmes temps réel sont de plus en plus omniprésents avec la prolifération des systèmes embarques, plus petits, plus puissants et peu couteux. Ces systèmes sont utilisés dans des domaines très variés, et font partie de notre vie de tous les jours. Ceci a rendu l'électronique de plus en plus présente sous plusieurs variétés de forme : les moyens de transport, les téléphones portables, le multimédia, et tout système embarqué.

Les systèmes embarqués deviennent de plus en plus complexes. C'est pour cette raison que les concepteurs ont recours à l'emploi d'un noyau temps réel multitâche (RTOS) pour gérer cette complexité.

Ce RTOS permet à une application d'être visualisée sous forme d'un ensemble de modules qui s'exécutent quasi-simultanément sur un même processeur. Le système temps réel doit également pouvoir prendre immédiatement en compte les stimuli synchrones provenant du matériel, il doit donc pouvoir gérer les interruptions. On associe classiquement à chaque interruption, un module appelé routine d'interruption (ISR), qui réalise l'interface entre l'application et son environnement matériel. Ces routines d'interruption, encore appelés séquences immédiates, ont pour but de répercuter l'occurrence d'un évènement *(seuil de température franchi, capteur activé, etc. ...)* à la tâche chargée de son traitement.

FreeRTOS est un système d'exploitation temps réel gratuit et (open source) développé par

***Real Time Engineers***, parmi ses fonctionnalités figurent les caractéristiques suivantes : des tâches préemptives, un support de plusieurs architectures de microcontrôleurs, un faible encombrement (4,3Ko sur un ARM7 après compilation3), écrit en C et compilé avec divers compilateurs . Il permet également un nombre illimité de tâches à exécuter en même temps et aucune limitation quant à leurs priorités tant que le matériel utilisé peut se le permettre.

Enfin, il implémente des files d'attente, des sémaphores binaires et de comptage et des mutex.

**Le noyau FreeRTOS**

Noyau RTOS leader sur le marché, standard de facto et multi-plateforme

Développé en partenariat avec les plus grands fabricants de puces du monde sur une période de 15 ans, FreeRTOS est un système d'exploitation en temps réel (RTOS) leader sur le marché pour les microcontrôleurs et les petits microprocesseurs. Distribué gratuitement sous la licence open source du MIT, FreeRTOS comprend un noyau et un ensemble croissant de bibliothèques pouvant être utilisées dans tous les secteurs industriels. Avec un téléchargement toutes les 175 secondes, FreeRTOS est construit en mettant l'accent sur la fiabilité, l'accessibilité et la facilité d'utilisation.

FreeRTOS est téléchargé toutes les 175 secondes (en moyenne, au cours de l'année 2018).

FreeRTOS a été classé premier de sa catégorie dans toutes les études de marché de l'EETimes Embedded Market Survey depuis 2011, première année où il a été inclus.

FreeRTOS offre des risques de projet et un coût total de possession plus faibles que les alternatives commerciales car : Il est entièrement pris en charge et documenté.

FreeRTOS est conçu pour être simple et facile à utiliser : Seuls 3 fichiers sources communs à tous les ports RTOS et un fichier source spécifique au microcontrôleur sont nécessaires, et son API est conçue pour être simple et intuitive.

**Pourquoi choisir FreeRTOS ?**

*"Il est probablement plus sûr de dire à ce stade que FreeRTOS est plus soumis à l'examen des pairs que tout autre RTOS disponible sur la planète. Je l'ai utilisé dans plusieurs projets, dont un environnement multiprocesseur qui utilisait plus de 64 processeurs et devait fonctionner de manière fiable pendant des mois. Le cœur du RTOS a bien fonctionné. Prenez FreeRTOS pour un tour".* - John Westmoreland

FreeRTOS est vraiment gratuit et pris en charge, même lorsqu'il est utilisé dans des applications commerciales. La licence open source FreeRTOS du MIT ne vous oblige pas à exposer votre propriété intellectuelle. Vous pouvez mettre un produit sur le marché en utilisant FreeRTOS sans payer des frais.

FreeRTOS...

Fournit une solution unique et indépendante pour de nombreuses architectures et outils de développement différents.

Dispose d'un minimum de ROM, de RAM et de frais de traitement. En général, une image binaire du noyau RTOS sera de l'ordre de 6 à 12K octets.

Est très simple - le noyau du noyau 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).

Contient un exemple préconfiguré pour chaque port. Pas besoin de savoir comment mettre en place un projet - il suffit de télécharger et de compiler !

Possède un excellent forum de support gratuit, contrôlé et actif.

Fournit une documentation abondante.

Est très évolutif, simple et facile à utiliser.

FreeRTOS offre une alternative de traitement en temps réel plus petite et plus facile comparativement à d’autres systèmes comme eCOS, Linux embarqué (ou Linux temps réel) et même uCLinux ne conviennent pas, ne sont pas appropriés ou ne sont pas disponibles.

Notre travail consistait à donner une description détaillée du noyau FreeRTOS en débutant par une revue sur les items de configuration des pré-exécutions, puis par le gestionnaire de tâches et le gestionnaire de listes et en détail sur le Scheduler et les mécanismes des files d'attente par le gestionnaire des files d'attente.

Et par la suite on a évoqué le principe général d'un débogueur de type « RTOS aware » en présentant quelques types de débogueurs temps réel tels que les probepoints et ReTis, ensuite la trace des appels système.

Comme environnement de développement, on a vu comment mettre en place ECLIPSE C/C++ ; installation et configuration pour s'adapter aux contraintes du RTOS.

On a vu la puissance de FreeRTOS+Trace comme outil de diagnostic pour les systèmes d'exploitation des systèmes embarqués basés sur FreeRTOS.

On a vu l'utilisation de FreeRTOS Win32 Simulator par la section qui décrit une application qui fonctionne dans le simulateur Win32 FreeRTOS, et pouvoir ainsi évaluer **FreeRTOS +CLI** et **FreeRTOS+Trace** sur un PC de bureau standard sans avoir de connecter un équipement externe.

Il est donc, possible de mette en place un environnement agréable et confortable pour procéder à la conception et à l'étude des systèmes embarqués vu le nombre , la simplicité et la puissance d'outils qui sont à notre disposition actuellement et la grande plage de microprocesseur et microcontrôleurs que ce système peut gérer avec un coût et un temps minimum ce qui rendra les concepteurs à accroitre leur productivité et le niveau de complexité des systèmes embarqués.

[1] RichardBarry : FreeRTOS "The Standard Solution For Small EmbeddedSystems".
http://www.freertos.org/

[2] M. Sahari PhD, K. Djahl, "FreeRTOS-Aware Débogueur pour Systèmes Embarqués" Embedded Systems Conference ESC09, École Militaire Polytechnique Alger, 5-6 Mai, 2009.

[3] PICos18 : Noyau temps réel pour PIC18
Tutorial & Manuel du Développeur.
www.picos18.com/Download/PICos18_tuto_fr.pdf

[4] Marc Rivaletto : Cours temps réel.
Système Temps Réel (IUP GEII Université de Pau).
www.web.univpau.fr/ENSEIGNEMENT/IUPGEII/Index/prof/rivaletto/tpsreel/coursstr.pdf

[5] Richard Barry : Multitasking on an AVR "Example C implementation of a multitasking kernel for the AVR , March 2004"
www.engineering.uakron.edu/grover/web/ee470/handouts/Free%20RTOS.pdf

[6] N.Navet,J.Thomesse : Equipe TRIO laboratoire LORIA à Nancy
L 'ordonnancement, la clé d'une gestion efficace des ressources.
www.loria.fr/Avnnavet/publi/jautomatise2002.pdf

[7] Samia Bouzcfranc : Maîtrc dc Conférences CEDRIC CNAM.
Introduction aux systèmes temps réel
www.cedric.cnam.fr/Nbouzefra/cours/cours-LS/Introduction.pdf

[8] Richard Barry : FreeRTOS Overview Set of slides is taken from the documentation existing on the FreeRTOS website
www.eng.uwi.tt/depts/elec/staff/feisal/ee33a/resources/2005-2006-d.pdf

[9] Jean J. Labrosse : MicroC/os-2, Second Edition
The Real-Time Kemel
Côte : 08-07-01 (bibliothèque génie électrique).

[10] Dr Dobb's : Debugging multi-threaded applications With RTOS-aware tools
http://www.embedded-control-europe.com/c-ece-knowhow/605/ecemay05p21.pdf

[11] Stelian Pop : Conférence de Stelian Pop sur La mise au point en espace noyau
http://popies.net/conferences/kernel-debugging.pdf

[12] Marion Strauss : Brique projet - T3 2006.
Extension d'un outil de trace pour système embarqué temps réel.
http://beru.univ-brest.fr/Nsinghoff/cheddar/contribs/examples-of-use/strauss06.pdf

[13] Stelian Pop : Solutions LINUX 2003
Débogage du noyau Linux
www.alcove.fr/IMG/pdf/kernel-debugging.pdf

[14] K. Andersson, R. Andersson: A comparison between FreeRTOS and RTLinux in embedded real-time systems.
http://www.streambag.se/files/rtproj.pdf

[15] Richard Barry : FreeRTOS Trace Hook Macros
"The Standa Solution for Small Embedded Systems".
www.freertos.org/rtos-trace-macros.html

[16] Brian Fellowes, MQX Embedded
Debugging multi-threaded applications with RTOS-aware tools.
www.embedded-control-europe.com/c-ece-knowhow/605/ecemay05p21.pdf

[17] Richard Barry: forum FreeRTOS.org
http://sourceforge.net/forum/forum.php?forum-id=382005

[18] Site officiel des développeurs de CrossView www.tasking.com

[19] Stelian Pop : Solutions LINUX 2003, Débogage du noyau Linux.
www.alcove.fr/IMG/pdf/kernel-debugging.pdf

[20] Mikaöl BRIDAY : Thöse de Doctorat de l'Université de Nantes (2004).
Validation par simulation fine d 'une architecture opérationnelle
www.irccyn.ec-nantes.fr/~briday/Briday-PhD-04.pdf

[21] Richard Barry: FreeRTOS Legacy Trace Utility
"The Standard Solution for Small Embedded Systems".
www.freertos.org/a00086.html

[22] Joint Test Action Group (JTAG) web page.
http://www.jtag.com/

[23] Cheddar a real time scheduling framework web page.
http://beru.univ-brest.fr/msinghoff/cheddar/ug/c,heddar-r2.html

[24] T. Pennello, "SeeCode : A New Approach to a Debugger", Technical Director of Software Tools, Ph.D. MetaWare Incorporated.

[25] S. Hung "New Tracing and Performance Analysis Techniques for Embedded Applications, " The 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2008.

[26] Dogan Ibrahim: Advanced PIC Microcontroller Projects in C.
From USB to RTOS with the PIC18F series, London, 2007.

[27] Aide officielle pour développer avec Eclipse en C/C++ :
http://help.eclipse.org/juno/topic/org.eclipse.cdt.doc.user/concepts/cdt_o_home.htm?cp=ll

[28] La page officielle CDT : http://www.eclipse.org/cdt/

[29] Le Wiki CDT : http://wiki.eclipse.org/CDT

[30] Les nouveautés de la version CDT 8.2 : http://wiki.eclipse.org/CDT/User/Newln82

[31] Les nouveautés de la version CDT 8.1 : http://wiki.eclipse.org/CDT/User/Newln81

[32] Site officiel de MinGW : http://mingw.org/

[33] Site officiel de Cygwin : http://cygwin.com/

[34] Les nouveautés de la version CDT 8.0 : http://wiki.eclipse.org/CDT/User/Newln80