REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Université Akli Mohand Oulhadj - Bouira –

Faculté des Sciences et Sciences Appliquées

Département de Génie Electrique

Mémoire de fin d'études

Présenté par :

Meue HARRACHE Sihem

Meue KHARRAF Amina

En vue de l'obtention du diplôme de **Master** en:

Filière : ELECTRONIQUE
Option : Electronique des Systèmes Embarqués

Thème :

La gestion de ressources matérielles dans un microcontrôleur par l'utilisation d'un système d'exploitation en temps réel RTOS

Devant le jury composé de :

Mme. AIT SAADI	MCA	UAMOB	Président
Mr. Medjedoub.I	MAA	UAMOB	Encadreur
Mr. BENZIANE.M	MCA	UAMOB	Examinateur
Mr. LABANJI .M		UAMOB	Co_Encadreur

Année Universitaire 2019/2020

Remerciement

*Tout d'abord nous remercions **ALLAH** le tout puissant, qui nous a donné la force et la patience d'accomplir ce modeste travail.*

*Nous tenons à remercier grandement notre Encadreur **Mr. Labandji** de nous avoir proposé ce sujet de mémoire et de l'attention qu'il a porté à notre travail.*

*Nous remerciant également **Mr. Madjdoub** chef de département Génie électrique pour sa grande disponibilité et ses précieux conseils.*

*Nous remercions chaleureusement **Mr. CHATBI Hamid** qui nous est conseille et encourage tout le long de notre travail. Nous leurs exprimons notre profonde gratitude pour leur attention, et surtout pour la grande patience qu'il est manifeste. Sans leur soutien et leur disponibilité, ce travail n'aurait pas pu être couronne de succès.*

Nous remerciant également tous les enseignants du département d'électronique de l'université de Bouira, plus spécialement les membres de jury de notre travail.

Nous remercions nos familles pour leurs soutiens et leurs encouragements et Spécialement nos chers parents qui ont toujours été là pour nous.

Nous voudrions exprimer notre reconnaissance envers les amis et collègues qui ont apporté leur soutien moral et intellectuel tout au long de notre démarche.

Et pour finir, Nous adressons nos sincères remerciements à tous ceux qui ont participé de près ou de loin à l'élaboration de ce travail.

DEDICACE

Je dédie ce mémoire :

À l'homme de ma vie, mon soutien moral et source de joie et de bonheur, celui qui s'est toujours sacrifié pour me voir réussir, à toi papa, Merci.

À la lumière de mes jours, la source de mes efforts, la flamme de mon cœur, ma vie et mon bonheur, maman que j'adore.
À mes chères sœurs et mon cher frère que dieu leur offre tout ils ce dont rêvent.

Je le dédie aussi ce mémoire à mon binôme AMNA et à toute ma famille.

Aux personnes qui m'ont toujours aidé et encouragé, qui étaient toujours à mes côtés, et qui m'ont accompagné durant mon parcours d'études supérieures, à mes aimables amis et collègues d'étude.

Et à tous ceux qui ont contribué de près ou de loin à l'élaboration de ce modeste travail, je vous dis merci.

SIHEM

DIDICAS

je dédie ce modeste travail accompagné d'un profond amour :

A ma maman qui ma soutenu durant ces années. Qu'elle trouve ici le témoignage de ma profonde reconnaissance.

A mon très chère père que dieu ait pitié de lui.

A tous mes chères frères et sœurs et mon fiancé et tous ceux qui ont partage avec moi tous les moments lors de la réalisation de ce travail.

A ma famille et mes amies proches, et spécialement mon oncle Karim qui m'a toujours encouragé durant ces années d'étude, je dédie ce travail à tous ceux qui ont participe à ma réussite.

Amina

Table des matières

Chapitre I : Les systèmes embarqués et le temps réel

Chapitre II : Les systèmes d'exploitation en temps réel

Chapitre III : FreeRTOS

Chapitre IV : Application et réalisation

Table des matières

Liste d'abréviation

API : Application Programming Interface.

ARM : Advanced Risc Machine.

CBD : Component Based Design.

CPU : Central Processing Unit.

E/S : Entre / Sortie.

FADO : Florida Association of Diving Operators.

FIFO : First In First Out.

IBM : International Business Machines.

ICSP : In Cicuit Serial Programming.

IDE : Integrated Development Environment.

IP : Internet Protocol.

IPC : Inter Processus Communication.

ISR : Interruption Service Routine.

LAN : Local Area Network.

MCU : Micro Controller Unit.

MIPS : Microprocessor without Interlocked Pipelined Stages.

MIL-STD : Military Standard.

MPU : Micro Processor Unit.

NASA : National Aeronautics and Space Administration.

OS : Operating System.

PC : Personnel computer.

POSIX : Portable Operating System Interface for Unix.

PWM : Pulse Width Modulation.

RAM : Random Access Memory.

RISC : Reduced Instruction Set Computing.

ROM : Read Only Memory.

RPC : Remote procedure call.

RTOS : Real Time Operating System.

RTS : Real Time System.

SMP : Symmetric Multiprocessing.

TCB : Transmission Control Protocol.

TCP : Transfert Control Protocol.

UART : Universal Asynchronous Reciever Transmitter.

UC : Unit Controller.

UML : Unified Modeling Language.

USB : Universal Serial Bus.

VRTX : Versatile Real-Time eXecutive

WAN : Wide Area Network.

Liste des figures

Liste des tableaux

Introduction général

Introduction général

Les systèmes électroniques sont de plus en plus présents dans la vie courante. Les ordinateurs et micro-ordinateurs sont des systèmes électroniques bien connus. Mais l'électronique se trouve maintenant embarqué dans de très nombreux objets usuels : les téléphones, les agendas électroniques, les voitures. Ces systèmes sont appelés systèmes embarqués. Ils prennent une place de plus en plus importante dans notre société, ils servent à contrôler, réguler des dispositifs électroniques grâce à des capteurs embarqués dans des robots, des véhicules spatiaux, etc. Ils sont souvent utilisés par le public dans la vie de tous les jours sans même qu'on ne s'en rende compte.

Pour qu'un système embarque fonctionne, des programmes informatiques doivent être installés sur le matériel du système. L'un de ces programmes qui est le système d'exploitation, son rôle est l'effectuation d'un certain nombre d'opérations logiques à fin d'assurer l'interactivité entre les autres programmes et avec les ressources matérielles du système comme le processeur, la mémoire, et les périphériques E/S. Dans le domaine des systèmes embarqués, la notion de temps réel est abondante. De ce fait, le système d'exploitation dit système d'exploitation en temps réel, doit traiter les données d'entre et fournir la sortie souhaite dans un délai stipule.

Système d'exploitation en temps réel RTOS, en général, fournit au concepteur les outils temps réel nécessaires pour la gestion des processus, la gestion des ressources, la communication inter-processus, la gestion des systèmes de fichier ...etc. Le développement de RTOS a augmenté rapidement au cours des dernières décennies. Les applications de ces systèmes sont considérées comme robustes et représentent toujours un défi pour les concepteurs. Ces systèmes doivent garantir des contraintes de timing satisfaisantes lors de l'exécution de tâches complexes [1].

Il existe un grand nombre de RTOS, 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 [2]. ***FreeRTOS*** a porté notre attention, un petit Scheduler temps réel offrant un certain nombre d'avantages.

Notre objectif se résume dans la phrase suivante : « l'utilisation d'un système d'exploitation en temps réel RTOS sous la plateforme Arduino pour la gestion des ressources

matériel (UAL Registres, Interruption, bus….etc.) Afin d'avoir une exécution parallèle de multitâches ».

Ce mémoire comporte quatre chapitres organisés comme suit:

- Dans le premier chapitre, nous évoquerons le système embarqué, ses caractéristiques, ainsi que son classification…etc. Puis, nous avons présenté l'un des aspects les plus importants dans le domaine des systèmes embarqués est le temps réel.
- Le deuxième chapitre, nous présenterons les systèmes d'exploitation en temps réel RTOS, la définition, leurs catégories, et l'architecteur…etc.
- Le troisième chapitre, nous présentons une classe de RTOS qui est le FreeRTOS.
- Le but du quatrième chapitre est d'appliquer, simuler et réaliser.

Chapitre I

Les systèmes embarqués et le temps réel

Chapitre I : Les systèmes embarqués et le temps réel

I. 1. Introduction

Le domaine des systèmes embarqués est en train très rapidement de devenir un domaine technologique incontournable qui intervient dans notre vie quotidienne à tous les niveaux (par exemple habitations, espaces personnels, hôpitaux, lieux de travail, voitures et transports en général, centres industriels).

On entend souvent parler de temps réel dès qu'on parle de système embarqué. En fait, un système embarqué doit généralement respecter des contraintes temporelles fortes. Le temps réel est un concept un peu vague, et largement utilisé dans de nombreux contextes.

Les systèmes embarqués en temps réel sont couramment utilisés dans notre vie quotidienne. Leurs applications vont du simple appareil domestique, tel que les machines à laver, les systèmes du chauffage central ou les barrières automatiques…etc. Ces systèmes se diffèrent par bien des aspects, comme l'utilisation, la taille, et la criticité, mais restent néanmoins tous caractérisés par des contraintes temporelles plus ou moins strictes. Leur développement nécessite des techniques spécialisées. L'analyse des besoins doit prendre en compte l'aspect temporel qui doit être précisé et vérifié. La conception doit faire face à l'architecture du système, la répartition des tâches et des processeurs, l'ordonnancement de tâches…etc. Le tout est soumis à des contraintes en relation avec le temps et les ressources[1].

Dans ce chapitre on va aborder tout d'abord la définition de système embarqué, ses caractéristiques, ainsi que son classification, la deuxième partie de ce chapitre va être consacrée sur les systèmes temps réel, leurs types, et on se focalisera principalement sur la conception du système temps réel.

I. 2. Définitions de système embarqué

Michæl Barr définit un système embarqué comme« une combinaison de matériel et de logiciel, et avec peut-être des additifs mécaniques ou autres composants. Qui ont pour but de réaliser une fonction dédiée » [3]. Ou la définition « Un système embarqué est un système électronique qui comprend un ou plusieurs microcontrôleurs configurés pour exécuter une application dédiée spécifique ». Pour mieux comprendre l'expression « système embarqué », considérez chaque mot séparément. Dans ce contexte, le mot embarqué signifie "un ordinateur est caché à l'intérieur donc on ne peut pas le voir". Le mot « système » fait référence au fait qu'il existe des éléments humains qui agissent de concert pour atteindre le but commun. Comme mentionné plus tard, les dispositifs d'entrée/ sortie caractérisent le système embarqué,

lui permettant d'interagir avec le monde réel [4]. Ces systèmes sont composés de plusieurs couches, comme l'illustre **la figureI.1**. La couche la plus abstraite est la couche de l'application logicielle, elle communique avec une couche logicielle de plus bas niveau qui représente le système d'exploitation. Ensuite, vient le réseau de communication matériel, puis les composants matériels [5]

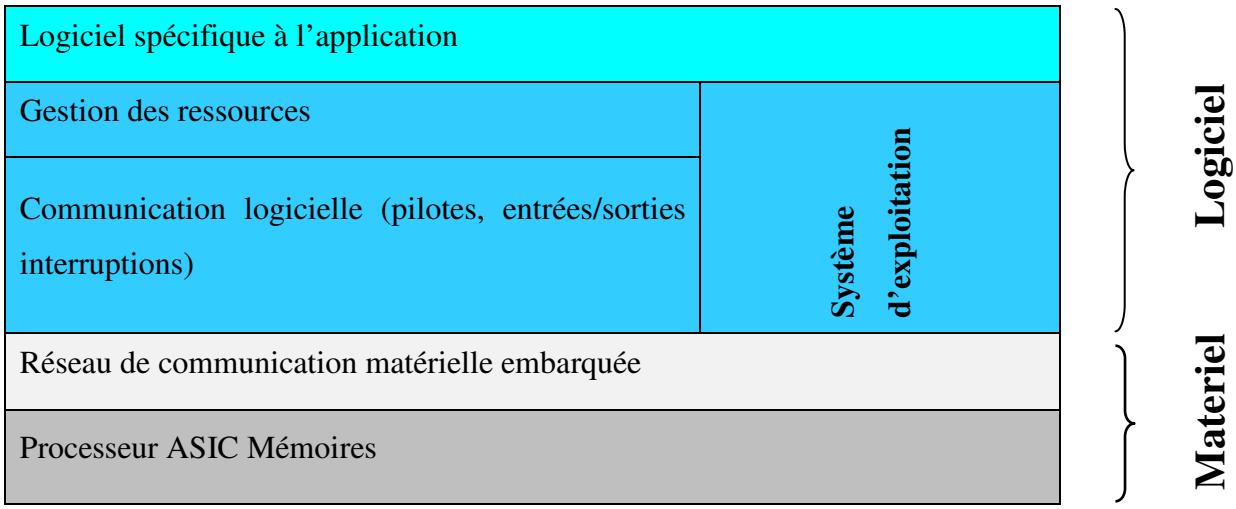


Figure 0.1. Présentation d'un système embarqué sous .forme de couches [5].

I. 3. Caractéristiques principales d'un système embarqué

Le logiciel qui contrôle le système est programmé ou fixé dans la mémoire flash ROM et n'est pas accessible à l'utilisateur de l'appareil. Même ainsi, la maintenance des logiciels reste extrêmement importante. La maintenance logicielle est la vérification du bon fonctionnement, les mises à jour, la correction des bugs, l'ajout de fonctionnalités et l'extension à de nouvelles configurations d'application et d'utilisateurs finaux. Les systèmes embarqués ont ces quatre caractéristiques [4]:

Premièrement, les systèmes embarqués remplissent généralement une seule fonction. Par conséquent, ils résolvent une gamme limitée de problèmes. Par exemple, le système intégré dans un four à micro-ondes peut être reconfiguré pour contrôler différentes versions du four dans une gamme de produits similaire. Mais, un four à micro-ondes sera toujours un four à micro-ondes, et vous ne pouvez pas le reprogrammer pour être un lave-vaisselle. Les systèmes embarqués sont uniques en raison des ports d'E / S du microcontrôleur auxquels les périphériques externes sont interfacés. Cela permet au système d'interagir avec le monde réel.

Deuxièmement, les systèmes embarqués sont fortement contraints. En règle générale, le système doit fonctionner selon des paramètres de performance très spécifiques. Si un système

embarqué ne peut pas fonctionner avec des spécifications, il est considéré comme un échec et ne sera pas vendu. Par exemple, un opérateur de téléphonie mobile obtient généralement 832 fréquences radio à utiliser dans une ville, un jeu vidéo portable doit coûter moins de 50 $, un régulateur de vitesse automobile doit faire fonctionner le véhicule à moins de 3 mi / h de la vitesse de consigne, et un lecteur MP3 portable doit fonctionner pendant 12 heures avec une charge de batterie.

Troisièmement, de nombreux systèmes embarqués doivent fonctionner en temps réel. Dans un système en temps réel, nous pouvons mettre une limite supérieure sur le temps nécessaire pour effectuer la séquence d'entrée-calcul-sortie. Un système en temps réel peut garantir une limite supérieure du pire des cas sur le temps de réponse entre le moment où les nouvelles informations d'entrée deviennent disponibles et lorsque ces informations sont traitées. Une autre exigence en temps réel qui existe dans de nombreux systèmes embarqués est l'exécution de tâches périodiques. Une tâche périodique est une tâche qui doit être effectuée à intervalles de temps égaux. Un système en temps réel peut mettre une petite limite sur l'erreur de temps entre le moment où une tâche doit être exécutée et le moment où elle est réellement exécutée. En raison de la nature en temps réel de ces systèmes, les microcontrôleurs de la famille TM4C disposent d'un riche ensemble de fonctionnalités pour couvrir tout le respect du temps.

La quatrième caractéristique des systèmes embarqués est leur faible mémoire requise par rapport aux ordinateurs à usage général. Il existe des exceptions à cette règle, telles que celles qui traitent la vidéo ou l'audio, mais la plupart ont des besoins en mémoire mesurés en milliers d'octets. Au fil des ans, la mémoire des systèmes embarqués a augmenté, mais l'écart de mémoire entre les systèmes embarqués et les ordinateurs à but restent. Les microcontrôleurs d'origine avaient des milliers d'octets de mémoire et le PC en avait des millions. Maintenant, les microcontrôleurs peuvent avoir des millions d'octets, mais le PC en a des milliards.

I. 4. Classification d'un système embarqué [6]

- **Système embarqué autonome :** Ce système ne nécessite pas de système hôte comme un système informatique, il fonctionne par lui-même. Il prend l'entrée des ports d'entrée analogiques ou numériques et traite, calcule et transfère les données et donne les données résultantes via l'appareil connecté - qui contrôle, pilote ou affiche les

appareils associés. Par exemple, les systèmes embarqués autonomes sont les lecteurs MP3, les appareils photo numériques, les consoles de jeux vidéo, les fours à micro-ondes et les systèmes de mesures de température.

- **Systèmes embarqués en temps réel :** Un système appelé système embarqué en temps réel, qui donne une sortie requise à un moment donné. Ces types de systèmes embarqués respectent les délais d'exécution d'une tâche. Les systèmes embarqués en temps réel sont classés en deux types tels que les systèmes embarqués en temps réel doux et les systèmes embarqués en temps réel dur basés sur la précision du temps.

- **Système embarqué en réseau :** Les systèmes embarqués en réseau sont liés à un réseau pour accéder aux ressources. Le réseau connecté peut être LAN, WAN ou Internet. La connexion peut être n'importe quelle connexion filaire ou sans fil. Ce type de système embarqué est le domaine technologique à la croissance la plus rapide dans les applications du système embarqué. Le serveur Web intégré est un type de système dans lequel tous les dispositifs intégrés sont connectés à un serveur Web et accédés et contrôlés par un navigateur Web. Par exemple, le système intégré en réseau LAN est un système de sécurité domestique dans lequel tous les capteurs sont connectés et fonctionnent sur le protocole protégé TCP / IP.

- **Systèmes embarqués mobiles :** Les systèmes embarqués mobiles sont hautement préférables dans les appareils embarqués portables comme les téléphones portables, les mobiles, les appareils photo numériques, les lecteurs mp3 sans fil et les assistants numériques personnels, etc. La limitation de base de ces appareils est les autres ressources et la limitation de la mémoire.

I. 5. Système en temps réel

I. 5. 1. Définition 1

Un système en temps réel est un système informatique qui doit satisfaire des contraintes de temps de réponse limités ou des risques de rupture, y compris une panne. Ce système dont l'exactitude logique est basée à la fois sur l'exactitude des résultats et sur leur actualité. Dans tous les cas, en rendant inutile la notion de rapidité, chaque système devient un système temps réel [7].

I. 5. 2. Définition 2

Un système temps réel est un système numérique qui se compose d'un ou plusieurs sous-systèmes devant répondre à un ensemble de stimuli provenant de l'environnement externe dans un intervalle de temps fini et spécifié dicté par ce même environnement afin de le contrôler .Une réponse hors échéance est invalide même si son contenu semble correct. L'absence de réponse est aussi grave qu'une réponse erronée, voire plus [8,9].

Les systèmes temps réel sont souvent des systèmes réactifs ou embarqués. Les systèmes réactifs sont ceux dans lesquels la gestion des tâches est motivée par une interaction continue avec leur environnement, par exemple, un système de Contrôle incendie réagit à certains boutons pressés par un pilote [7].

I. 6. Conception des systèmes temps réel [10,11]

I. 6. 1. Architecture physique

L'architecture physique comme son nom l'indique définir l'architecture matérielle du système, prise en charge généralement par un groupe de concepteurs matériels. Contrairement à la conception logicielle qui est prise par un tout autre groupe de concepteurs logiciels. Pour une bonne conception du système, les deux groupes doivent être en relation constante l'un avec l'autre tout au long du cycle de développement, en raison de prendre des décisions communes sur les différents points qui impliquent les deux parties. L'architecture physique a pour rôle de définir les composants matériels, comme les unités de calcul, les mémoires, les supports de communication, les capteurs... et leurs distributions spatiales. Une architecture générique pour un système temps réel est décrite dans **la figure.I.2**. Cette architecture est un modèle de tout système fondé sur l'interaction avec un environnement extérieur par l'intermédiaire des capteurs et des actionneurs.

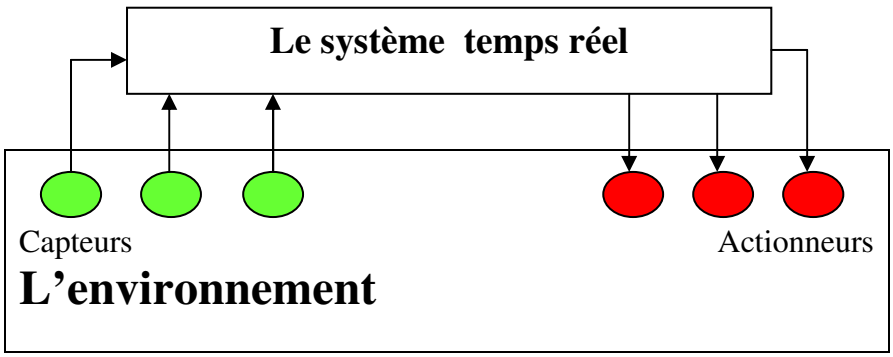


Figure 0.2. Architecture générique d'un système temps réel [10].

Le système temps réel le plus simpliste est un système avec un processeur unique, mais dans de nombreux cas, les systèmes temps réel sont des systèmes informatiques distribués composés d'un ensemble de processeurs interconnectés par un réseau de communication. Plusieurs raisons peuvent pousser les concepteurs à construire un système temps réel distribué, parmi ceux-ci :

- La distribution physique de l'application.
- Les exigences de calcul qui peuvent ne pas être facilement assuré par un seul processeur.
- Le besoin de redondance pour répondre à la disponibilité, la fiabilité, ou d'autres exigences en matière de sécurité.
- Afin de réduire le câblage du système.

I. 6. 2.Modèles d'interaction

Un modèle d'interaction décrit les règles par lesquelles les composants interagissent les uns avec les autres (dans cette section, nous allons utiliser le terme composant pour désigner une entité logicielle cohérente, comme par exemple une tâche ou un module). Le modèle d'interaction régit bien les flux de contrôles que les flux de données entre les composants du système. Pour la conception logicielle, le modèle d'interaction compte parmi les plus importantes décisions que les concepteurs logiciels doivent prendre. Malheureusement cette décision est souvent prise implicitement, du fait que le modèle d'interaction est souvent implémenté par le système d'exploitation ou le middleware choisi pour le système.

Lors de la conception d'un système temps réel, l'attention devrait être accordée à propriétés temporelles du modèle d'interaction choisie. Certains modèles ont un comportement temporel plus prévisible et plus robuste que d'autres. On peut citer par exemple quelques modèles des plus prévisibles et les plus utilisés dans la conception des systèmes temps réel comme le modèle tuyaux et filtres, le modèle éditeur-abonné, et le modèle tableau noir.

D'un autre côté, il existe des modèles d'interaction qui augmentent l'imprécision temporelle du système. En connaissance de cause, ces modèles doivent être pris avec précaution, et si possible, être évités lors de la conception d'un système temps réel. Les deux plus importants, et communément utilisés sont le modèle clients-serveur et le modèle boîte aux lettres.

- **Le modèle tuyaux et filtres** : Dans ce modèle, les flux de données et les flux de contrôle sont spécifiés en utilisant les ports d'entrée et de sortie des composants. Un composant devient apte pour l'exécution que lorsque les données sont arrivées sur ses ports d'entrées et il termine son exécution par produire ces flux sur ses ports de sortie.

  Ce modèle convient très bien pour de nombreux types de systèmes de contrôle, car les règles de contrôle sont facilement implémentable sur ce type-modèle. Ainsi, il a largement été utilisé dans la conception temps réel. Le modèle présente de bon caractéristiques temps réel, en sachant que les deux flux de données et de contrôle d'une façon unidirectionnelle traversent une série de composants, l'ordre d'exécution et les délais de bout en bout sont facilement calculables, ainsi le modèle devient facilement prévisible dans le temps.
- **Le modèle éditeur-abonné** : Se base sur le principe qu'un éditeur produit des publications sous forme de valeurs de données ou de contrôles, que les abonnés sont disposés à les utiliser. Le modèle éditeur-abonné est plus ou moins similaire au modèle tuyaux et filtres, quoique généralement le modèle éditeur-abonné dissocie le flux de données du flux de contrôle. Aussi, un abonné a généralement l'opportunité de choisir délibérément le déclenchement de son exécution parmi les différentes publications de son éditeur. Si on prend le cas où l'abonné choisit de se déclencher à chaque nouvelle publication, le modèle prend exactement la forme du modèle tuyaux et filtres. En plus de choisir la publication, un abonné a le droit d'ignorer la date de la publication et de prendre que la dernière valeur publiée. Aussi, pour le modèle éditeur-abonné, les éditeurs ne sont pas nécessairement conscients de l'identité, ni même de l'existence de leurs abonnés. Cela donne une plus grande abstraction dans la conception du système temps réel.

  Comme pour le modèle tuyaux et filtres, le modèle éditeur-abonné offre de bonnes propriétés temporelles. Toutefois, une condition préalable pour pouvoir faire l'analyse des systèmes utilisant ce modèle, est que les composants abonnés doivent indiquer d'une façon explicite aux éditeurs les valeurs auxquelles ils sont abonnées (ce qui n'est pas imposé par le modèle lui-même), cette information sert par exemple dans un système embarqué de décider des valeurs qui doivent être publiées sur son réseau de communication, et les nœuds qui doivent réceptionner ces valeurs.

- **Le modèle tableau noir:** Le modèle du tableau noir permet à des variables globales d'être publiées sur une zone mémoire accessible pour tous les composants (le tableau noir fait référence à la métaphore d'un tableau d'une classe d'étude où tout élève peut écrire dessus). Ainsi, le modèle permet à tout composant de lire ou d'écrire dans les valeurs des variables sur le tableau. De toute évidence l'utilisation de ce modèle dans la conception temps réel reste discutable, néanmoins c'est un modèle couramment utilisé, car dans certains cas, il fournit des solutions pragmatiques aux problèmes qui sont difficilement abordables avec les modèles d'interaction cités précédemment.
- **Le modèle client-serveur** : Dans le modèle client-serveur, un client invoque les services d'un serveur d'une manière asynchrone. Lors de l'invocation du service par le client un flux de contrôle (en plus d'un flux de données) est passé au serveur, et le contrôle reste dans le serveur jusqu'à ce qu'il termine son exécution. Entre temps le client reste bloqué jusqu'à ce que le serveur termine en lui renvoyant le flux de contrôle (et le flux de données). Ainsi le client peut continuer son exécution.

  Le modèle client-serveur est intrinsèquement imprévisible dans le temps, étant donné que les services sont invoqués de façon asynchrone, il est très difficile d'évaluer a priori la charge sur le serveur pour un service donné. Ainsi, il est difficile d'estimer le retard de l'invocation du service et, à son tour, il est difficile d'estimer le temps de réponse du client. Cette situation est encore plus compliquée par le fait que la plupart des composants se comportent souvent en tant que clients et serveurs (un serveur utilise souvent d'autres serveurs pour mettre en œuvre ses propres services). De ce fait le chemin du flux de contrôle et l'analyse temporelle du système sont très complexes à calculer.
- **Le modèle boîte aux lettres** : Un composant peut avoir un ensemble de boîtes aux lettres, et les composants communiquent les uns avec les autres en envoyant des messages dans les boîtes aux lettres des autres composants. Les messages sont généralement traités selon la stratégie première arrivée, premier servi (FIFO), ou par ordre de priorité (si l'expéditeur spécifie une priorité). Faire passer un message pour un expéditeur ne change pas sous le flux de contrôle. Tandis qu'un composant qui tente de recevoir un message qui n'est pas encore délivré par l'expéditeur peut se

bloquer jusqu'à ce que le message arrive (souvent, le récepteur spécifie un délai pour éviter le blocage pour une durée indéterminée).

Du point de vue de l'expéditeur, le modèle boîte aux lettres a les mêmes problèmes que le modèle client-serveur. Les données envoyées par l'expéditeur (et l'action que l'expéditeur s'attend du récepteur à effectuer) peuvent être retardées de façon imprévisible lorsque le récepteur est trop chargé. Aussi, en ce qui concerne la nature asynchrone du passage des messages, il est difficile de prévoir la charge d'un récepteur à un moment donné.

De plus, à partir du point de vue récepteur, la lecture des boîtes de messages est imprévisible dans le sens que le récepteur peut ou ne peut pas se bloquer sur la boîte de message (selon la présence ou non d'un message). Sans oublier que les boîtes aux lettres sont de taille limitée, il y a toujours un risque qu'un récepteur surchargé de messages perd des messages (donc une autre source d'imprévisibilité).

Les paradigmes d'exécution temporelle sont utilisés dans de nombreux systèmes embarqués critiques avec de hautes exigences de fiabilité (comme les systèmes de contrôle avionique), alors que la majorité des autres systèmes utilisent le paradigme événementiel. La fiabilité peut être aussi garantie par les paradigmes événementiels, mais en raison de l'observabilité fournie par les paradigmes temporels (la vérification des capteurs expliquée auparavant), la plupart des experts plaident pour l'utilisation de ce dernier dans les systèmes à haute exigence de fiabilité. Quoique son principal point faible soit son manque de souplesse et le surplus en consommation du processeur par rapport au paradigme événementiel.

I. 6. 3 Conception basée composant

Conception basée composant La conception basée composant ou Component-Based Design (CBD) est une approche intéressante pour le génie logiciel en général, et pour l'ingénierie des systèmes temps réel en particulier. Dans le Component-Based Design, un composant logiciel est utilisé pour encapsuler les fonctionnalités, et une fonctionnalité est uniquement accessible via l'interface du composant. Un système est composé par le regroupement d'un ensemble de composantes et des connexions de leurs interfaces.

La raison qui pourrait exposer l'utilité de la conception CBD pour les systèmes temps réel est la possibilité d'étendre les composants avec des interfaces d'introspection. Une interface

d'introspection ne fournit pas de fonctionnalité en soi, mais cette interface peut être très utile pour la récupération d'informations sur les propriétés extra fonctionnelles d'un composant. Extra-fonctionnelle veut dire les attributs caractéristiques du composant tel que les attributs qui fournissent la consommation de mémoire, le temps d'exécution, la période d'exécution d'une tâche…etc. Ces informations sont d'une extrême importance pour un système temps réel.

I. 6. 4. Outils pour la conception temps réel

L'un des outils les plus utilisés pour la conception de logiciels de nos jours est l'UML. Toutefois, UML est centré principalement sur les solutions client-serveur (puisqu'il est à base d'objets, et que le principal moyen de communication des objets est l'invocation de méthodes).Ce qui le rend peu sollicité pour la conception des systèmes temps réel. Par conséquent, des outils basés sur l'UML étendu pour inclure la conception temps réel ont vu le jour. Les deux plus connus sont des produits d'IBM: Rational Rose RealTime, et TelelogicRhapsody. Ces outils fournissent le support d'UML, avec des extensions pour le temps réel, tout en donnant aux concepteurs des modèles d'abstraction et de calcul.

I.7. Conclusion

Dans ce chapitre nous avons présenté l'un des systèmes électroniques et informatiques autonomes, qui sont étudiés pour effectuer des tâches précises, qui sont les systèmes embarqués et le temps réel. Dans le première partie, on a donné une perception introductive aux systèmes embarqués et le temps réel avec quelques définitions de système embarqué, ses caractéristiques principales tels que le remplissage d'une seule fonction, le fonctionnement en temps réel ... .Ainsi que son classification. Ensuite, on a défini le système en temps réel. Le reste de ce chapitre se focalise principalement sur les aspects fondamentaux à prendre en considération dans la conception des systèmes temps réel comme l'architecture physique, les modèles d'interaction, la conception basée composant , et on a fini par les outils pour la conception temps réel .

Ce qu'on peut retenir de ce chapitre que le terme « système embarqué » est utilisé pour désigner un système électronique conçu pour remplir une fonction dédiée et souvent intégrée dans un système plus vaste qui a réussi à faire converger plusieurs disciplines à première vue totalement disjoints pour former une technique des plus utilisées de nos jours.

*Chapitre **II***

Les systèmes d'exploitation en temps réel

II. 1. Introduction

Pour qu'un système embarqué soit capable d'accomplir ses fonctionnalités, des programmes informatiques doivent être installés sur le matériel du système. L'un de ces programmes, qui est le système d'exploitation. Un système d'exploitation [12](*Operating System* ou OS) est un ensemble de programmes spécialisés qui permet l'utilisation des ressources matérielles d'un ou plusieurs ordinateurs. Il assure le démarrage de l'ordinateur et l'exécution des logiciels applicatifs. Il remplit deux fonctions majeures : d'une part, la gestion des ressources matérielles (la mémoire, le processeur et les périphériques), en répartissant leur utilisation entre les différents logiciels ; d'autre part, la fourniture de services aux applications, en offrant une interface de plus haut niveau que celle de la machine physique. Cette interface présente la vision d'une (machine virtuelle), fournissant un ensemble de fonctions de base (appels système) pour l'écriture des applications.

Le recours accru à la technologie pour exécuter des tâches cruciales a conduit au développement des systèmes d'exploitation performants et déterministes, y compris des systèmes d'exploitation en temps réel (RTOS) [12].

Dans ce chapitre, nous allons entamer notre travail par la présentation des systèmes d'exploitation en temps réel, leur architecteur, ainsi qu'une analyse sur les principaux outils qui y régissent comme le Scheduler temps réel et les composants de RTOS. À la fin du chapitre, nous donnons quelques systèmes d'exploitation temps réel actuel, ainsi qu'une les facteurs essentiels que vous devez prendre en compte pour sélectionner RTOS.

II.2.Systèmes d'exploitation en temps réel

II.2 .1. Définition 1

Les systèmes d'exploitation (tels que Windows, Linux et MacOs) ont été créés pour fournir un environnement de programmation cohérent qui élimine le matériel sous-jacent pour faciliter l'écriture et la maintenance des programmes informatiques. Ils fournissent au programmeur d'applications de nombreuses primitives différentes (telles que comme threads et mutex) qui peuvent être utilisées pour créer des comportements plus complexes [13].

Tout système d'exploitation qui fournit une manière déterministe d'exécuter un morceau de code donné peut être considéré comme un système d'exploitation en temps réel. Cette définition de RTOS couvre un assez grand nombre de systèmes [13].

II.2 .2. Définition 2

Un système d'exploitation temps réel (RTOS : Real Time Operating System) est un système d'exploitation multi-tâche destiné aux applications temps réel. Il fournit le support de base pour la planification des tâches (scheduling), la gestion des ressources, la synchronisation et communication inter-processus, la gestion des entrées/sorties...etc. La différence avec un système d'exploitation non temps réel, c'est qu'il fournit en plus des services similaires à ce dernier, des services adaptés pour le développement des applications temps réel et un support pour le portage de l'ensemble système/applications sur le matériel du système embarqué [14].

Les différentes gammes d'applications RTOS sont généralement regroupées en trois catégories [13]:

- **Un système en temps réel dur** : Dans Hard RTOS doit respecter son échéance 100% du temps. Si le système ne respecte pas un délai, il est considéré comme ayant échoué. Cela ne signifie pas nécessairement qu'une panne blessera quelqu'un si elle se produit dans un système en temps réel dur - seulement que le système a échoué s'il manque une seule échéance.
- **Un système temps réel ferme** : Contrairement aux systèmes en temps réel dur, les systèmes en temps réel des entreprises doivent respecter leurs délais presque tout le temps. Si la vidéo et l'audio perdent momentanément la synchronisation, cela ne sera probablement pas considéré comme une défaillance du système.
- **Un système temps réel souple :** sont les plus laxistes en ce qui concerne la fréquence à laquelle le système doit respecter ses délais. Ces systèmes n'offrent souvent qu'une promesse de meilleur effort pour respecter les délais.

II.3.Architecteur des RTOS

L'architecture d'un RTOS dépend de la complexité de son déploiement. Les bons RTOS sont modulables pour répondre à différents ensembles d'exigences pour différentes applications. Pour les applications simples, un RTOS ne comprend généralement qu'un noyau. Pour les systèmes embarqués plus complexes, un RTOS peut être une combinaison de

différents modules, y compris le noyau, les piles de protocoles de réseau et d'autres composants, comme l'illustre **la figure.II.1** [15].

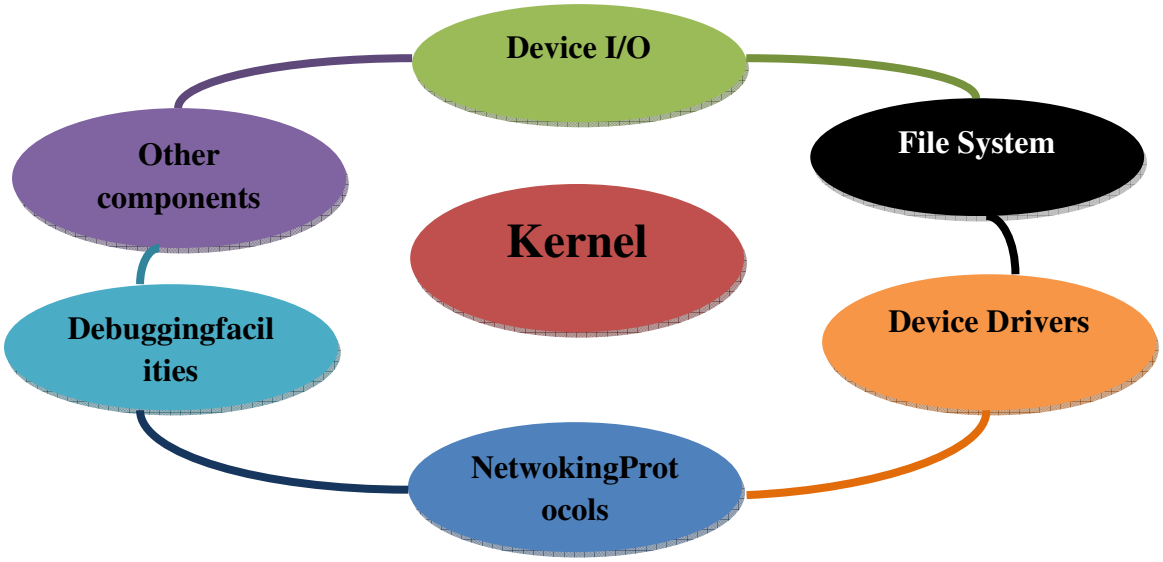


Figure .II. 1. Illustrant l'architecture générale de RTOS [15]

II.3.1.Noyau(Kernel)

Un système d'exploitation se compose généralement de deux parties : l'espace du noyau (mode noyau) et l'espace utilisateur (mode utilisateur). Le noyau est la plus petite et la plus centrale des composantes d'un système d'exploitation. Ses services comprennent la gestion de la mémoire et des périphériques, mais aussi de fournir une interface pour les applications logicielles afin d'utiliser les ressources. Des services supplémentaires tels que la gestion de la protection de programmes et le multitâche peuvent être inclus selon l'architecture du système d'exploitation. Il existe trois grandes catégories de modèles de noyaux disponibles, à savoir :

- **Noyau monolithique *(*Monolithic kernel)**

Il exécute tous les services systèmes de base (c'est-à-dire la gestion des processus et de la mémoire, la gestion des interruptions et la communication E/S, le système de fichiers, etc.) dans l'espace du noyau. En tant que tels, les noyaux monolithiques fournissent des abstractions riches et puissantes du matériel sous-jacent. Le nombre de commutations de contexte et de messages impliqués est considérablement réduit, ce qui le rend plus rapide que le micro-noyau (**figure.II.2**). Les exemples sont Linux et Windows.

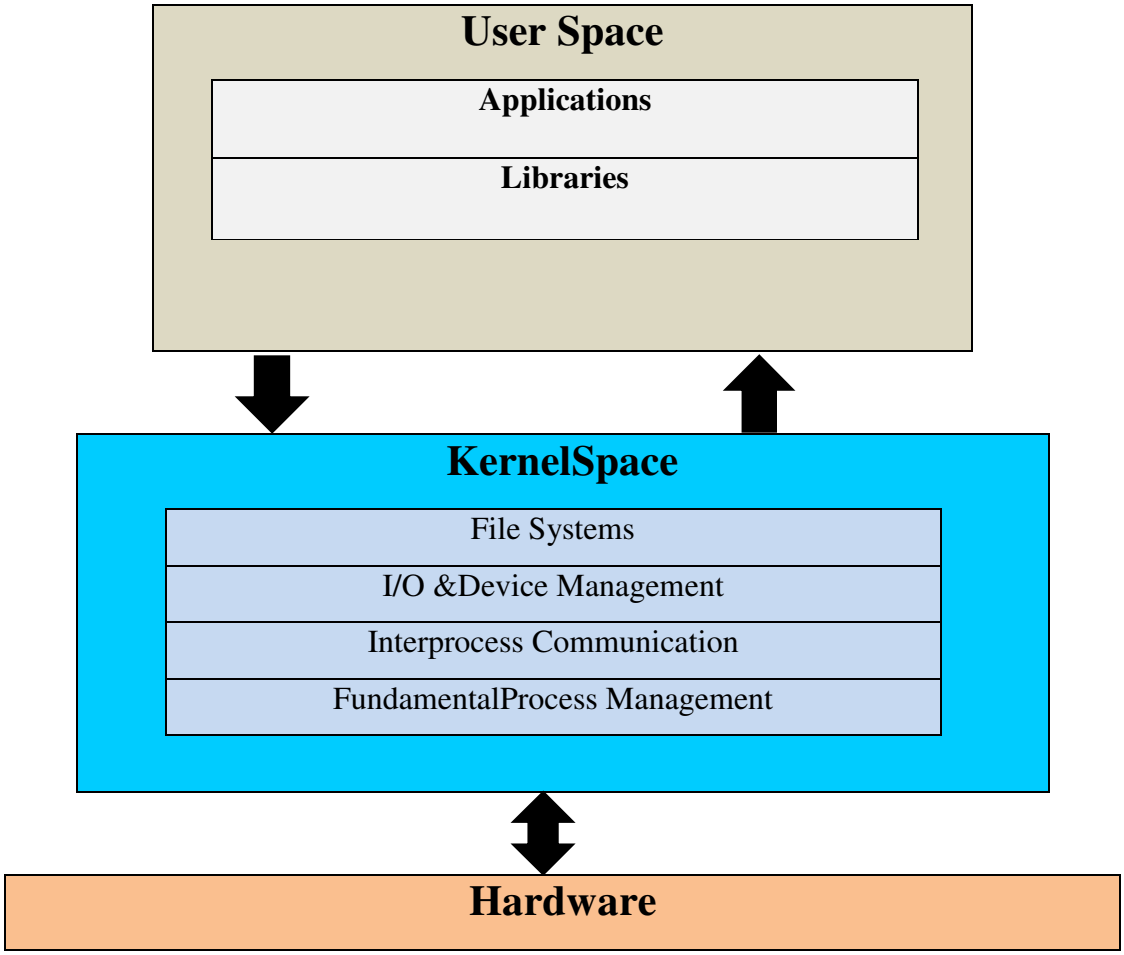


Figure . Ⅱ . 2. Système d'exploitation basé sur un noyau monolithique [15]

- **Micro-noyau (Microkernel)**

Il ne gère que la communication de base des processus (messagerie) et le contrôle des entrées/sorties. Les autres services du système (système de fichiers. Networking, etc.) résident dans l'espace utilisateur sous forme de démons/serveurs. Ainsi, les micro-noyaux fournissent un ensemble plus restreint d'abstractions matérielles simples. Il est plus stable que monolithique car le noyau n'est pas affecté même si les serveurs sont défaillants (c'est-à-dire le système de fichiers)**(figure.II.3)**. Les exemples sont AmigaOS et QNX**.**

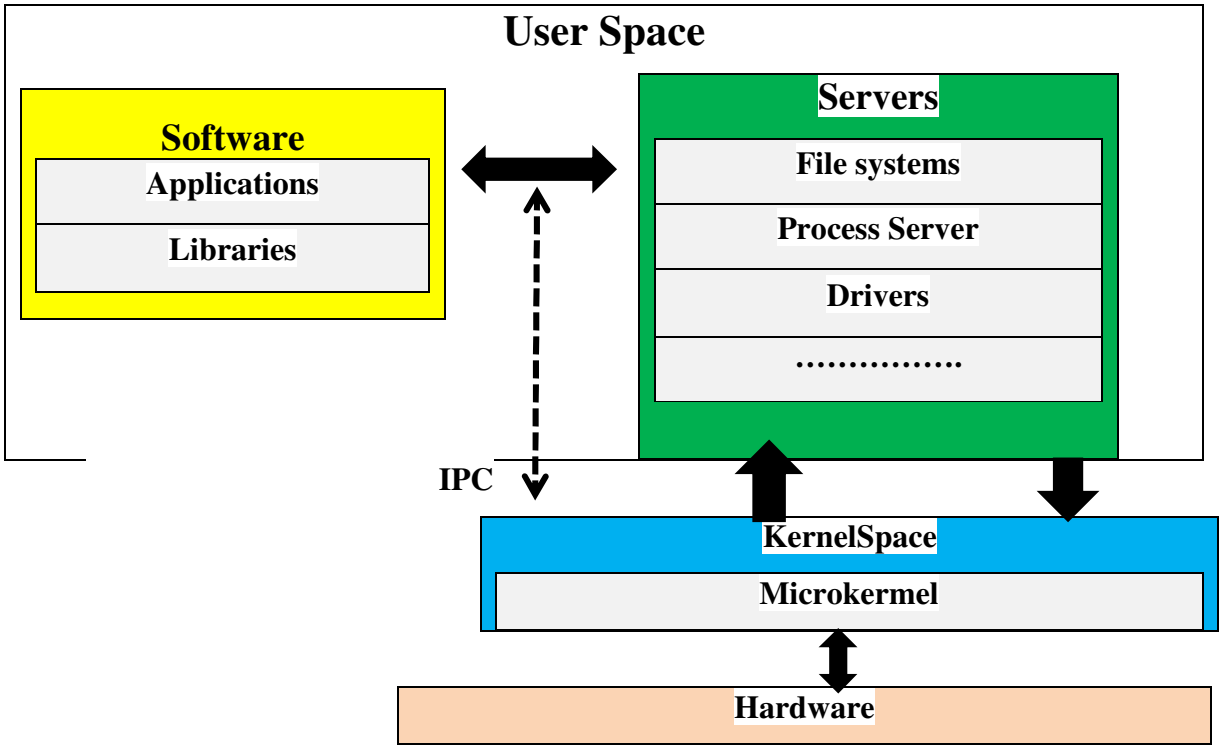


Figure .II. 3. Système d'exploitation basé sur les micro-noyaux [15].

- **Exokernel**

Le concept est orthogonal à celui des noyaux micro ou monolithiques en donnant à une application un contrôle efficace sur le matériel. Il ne fait fonctionner que des services protégeant les ressources (c'est-à-dire en suivant la propriété, en surveillant l'utilisation, en révoquant) en fournissant une interface de bas niveau pour les systèmes d'exploitation des bibliothèques (libOSes) et en laissant la gestion à l'application**(figure.II.4)**

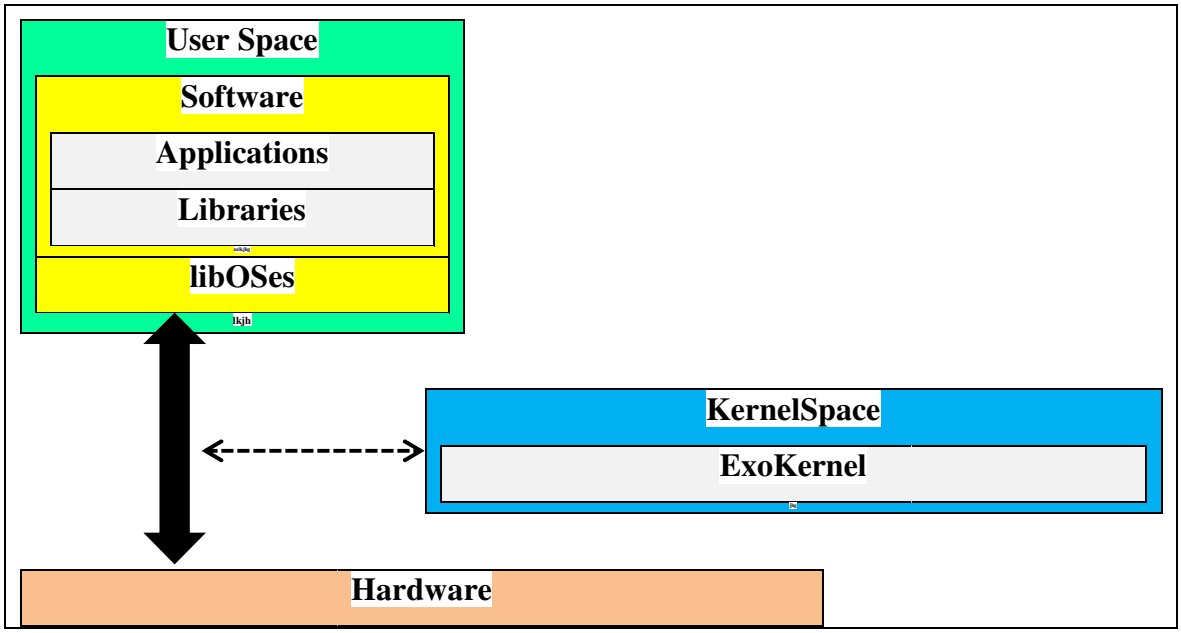


Figure .II. 4. Système d'exploitation basé sur exokernel [15].

Un RTOS évite généralement d'implémenter le noyau comme un grand programme monolithique. Le noyau est plutôt développé comme un micro-noyau avec des fonctionnalités configurables supplémentaires. Cette mise en œuvre offre l'avantage d'accroître la configurabilité du système, car chaque application embarquée nécessite un ensemble spécifique de services système en fonction de ses caractéristiques. Le noyau d'un RTOS fournit une couche d'abstraction entre le logiciel d'application et le matériel. Cette abstraction comprend six principaux types de services communs fournis par le noyau au logiciel d'application. **La figure.II.5** montre les six services communs d'un noyau RTOS.

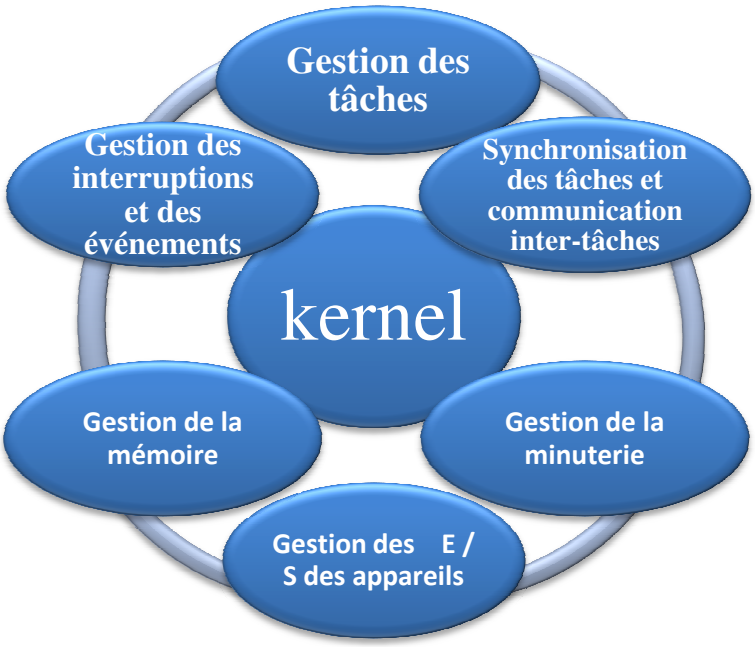


Figure .II. 5. Services du noyau RTOS.

II.3. 2 .Gestion des tâches

- **Tâches**

Une application en temps réel effectue un ensemble d'actions prédéfinies dans un certain laps de temps. Chacune de ces actions est généralement définie comme une tâche. Une tâche en temps réel peut être classée comme périodique ou apériodique selon son modèle d'arrivée. Les tâches avec des heures d'arrivée régulières sont appelées périodiques et les tâches avec des heures d'arrivée irrégulières sont appelées apériodiques. Chaque tâche a son propre contexte, qui contient l'environnement du processeur et les ressources système que la tâche voit chaque fois qu'elle est planifiée pour s'exécuter. Dans un système d'exploitation en temps réel, le contexte d'une tâche est stocké dans une structure de données, appelée le bloc de contrôle de tâche (TCB). Dans un algorithme d'ordonnancement préemptif, le contexte de chaque tâche est stocké ou rechargé à partir de la tâche TCB pendant le changement de contexte [16].

Les systèmes d'exploitation en temps réel conservent l'état actuel de chaque tâche du système. Les états peuvent avoir des noms différents dans différents systèmes d'exploitation et certains états supplémentaires peuvent exister dans certains systèmes d'exploitation. En règle générale, une tâche à tout moment peut être dans l'un des états suivants: en cours d'exécution, prêt, en attente et suspendre. Lors de leur création initiale, les tâches passent à l'état suspendu. L'activation est requise pour qu'une tâche créée entre dans son état prêt. L'état prêt indique qu'une tâche n'attend aucune ressource autre que la CPU. Selon différents algorithmes de planification, une tâche à l'état prêt peut être exécutée et entrer dans son état en cours d'exécution. Une seule tâche par processeur peut être en état d'exécution à tout instant: c'est la tâche qui utilise actuellement le processeur. Si une tâche est bloquée en raison de l'indisponibilité de ressources autres que la CPU, elle est mise en attente**.** ***La figure.II.6*** montre la transition de l'état des tâches dans les systèmes d'exploitation en temps réel [16].

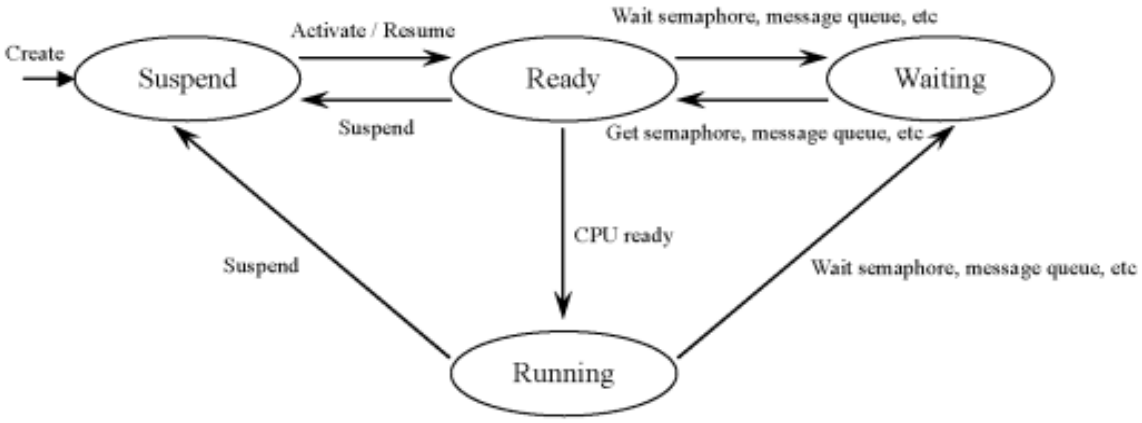


Figure .II. 6. Transition d'état de tâche**.**

- **Planificateur(Scheduler)**

Le planificateur enregistre l'état de chaque tâche et sélectionne parmi celles-ci celles qui sont prêtes à être exécutées et attribue le CPU à l'un d'entre eux. Un planificateur permet de maximiser l'utilisation de l'UC entre les différentes tâches d'un programme multitâche et deminimiser le temps d'attente [15]. Il existe deux types de mécanismes de planification de base pour les tâches en temps réel [16]: non préemptif ordonnancement et ordonnancement préemptif, En ordonnancement non préemptif, une fois qu'une tâche a commencé à s'exécuter, elle termine son exécution sans interruption (**La figure.II.7**donne un exemple d'ordonnancement non préemptif). Le principal avantage de la planification non préemptive est qu'elle a moins de surcharge de planification en raison de moins d'occurrences de changement de contexte. Cependant, il offre une planification plus faible. Inversement, dans la planification préemptive, l'exécution d'une tâche peut être préemptée par des tâches de priorité supérieure. À tout moment, la tâche prête à la priorité la plus élevée est en cours d'exécution. Une fois que toutes les tâches prêtes à priorité plus élevée ont terminé leur exécution, la tâche préemptée peut être reprise **(La figure.II.8** donne un exemple d'ordonnancement préemptif). Par rapport à la planification non préemptive, la planification préemptive peut fournir un degré plus élevé de planification en respectant d'abord les délais des tâches plus prioritaires. L'inconvénient de la planification préemptive est qu'elle nécessite une surcharge de planification plus élevée en raison d'un plus grand niveau de commutation de contexte.

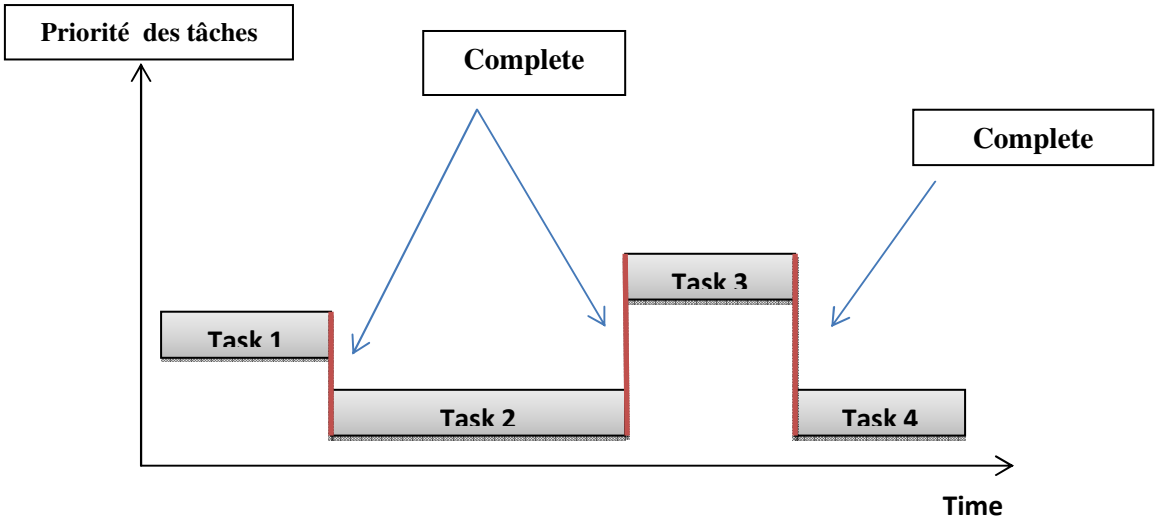


Figure .II. 7. Non-preemptive Scheduling [16].

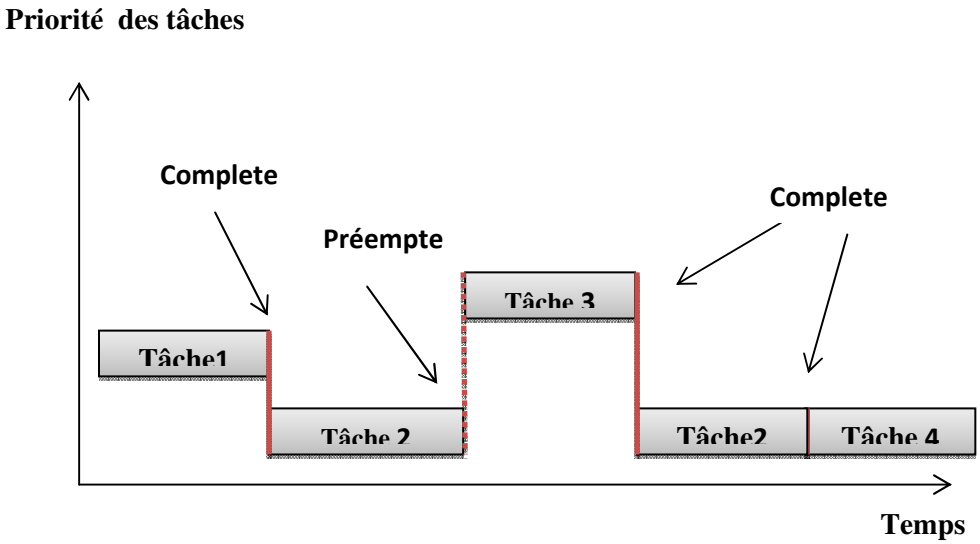


Figure . Ⅱ . 8: Preemptive Scheduling [16].

- **Dispatcher**

Le dispatcher donne le contrôle de la CPU à la tâche sélectionnée par le planificateur en effectuant une commutation de contexte et change le flux d'exécution. À tout moment où un RTOS est en cours d'exécution, le flux d'exécution passe par l'une des trois zones suivantes: via le code du programme de tâche, via une routine de service d'interruption ou via le noyau.

II.3.3.Synchronisation des tâches et communication inter-tâches

La synchronisation des tâches et les communications entre les tâches permettent de transmettre en toute sécurité les informations d'une tâche à une autre. Le service permet également aux tâches de se coordonner et de coopérer entre elles [17].

- **Synchronisation des tâches**

Un processus, également appelé thread ou tâche, est un programme simple qui pense qu'il a le CPU pour lui tout seul. Se référant à une tâche ou un processus dans un système d'exploitation en temps réel, aura la même signification. Si plusieurs tâches sont exécutées simultanément, cela s'appelle le multitâche. Les processus s'exécutent dans une boucle infinie et exécutent son code de programme étape par étape. Le système étant divisé en plusieurs processus, une synchronisation entre eux est nécessaire. La synchronisation peut se faire en plusieurs façons. Si la synchronisation implique plusieurs événements, elle peut être effectuée avec deux méthodes différentes. La synchronisation peut se produire si l'un des événements s'est produit, c'est ce qu'on appelle la synchronisation disjonctive. L'autre est conjonctif, maintenant tous les événements doivent avoir eu lieu pour être synchronisés et plus l'exécution peut commencer.

La synchronisation des tâches est réalisée à l'aide de deux types de mécanismes :

- Objets d'événement.
- Sémaphores.

Les objets événement sont utilisés lorsque la synchronisation des tâches est requise sans partage de ressources. Ils permettent à une ou plusieurs tâches d'attendre qu'un événement spécifié se produise. Un objet événement peut exister dans l'un des deux états: déclenché et non déclenché. Un objet événement dans un état déclenché indique qu'une tâche en attente peut reprendre. En revanche, si l'objet événement est dans un état non déclenché, une tâche en attente devra rester suspendue.

Figure.II .9.Principe de fonctionnement des objets d'évènement [17].

Le partage des ressources entre les tâches peut être un problème lorsque la coordination n'est pas bien faite. Par exemple, si la tâche A commence à lire un ensemble de données en cours de mise à jour par la tâche B, la tâche A peut recevoir des données corrompues - mélange de données nouvelles et existantes. Pour résoudre ce problème, les noyaux RTOS fournissent un objet sémaphore et des services de sémaphore associés pour garantir l'intégrité des données.

Un sémaphore est une variable protégée (ou type de données abstrait) et constitue la méthode classique pour restreindre l'accès aux ressources partagées (par exemple le stockage) dans un environnement multiprocesseur. Ils ont été inventés par **Edsger Dijkstra** au milieu des années 1960 et utilisés pour la première fois dans le T.H.E.

Un sémaphore a un nombre de ressources associé et une file d'attente. Le nombre de ressources indique la disponibilité de la ressource.

Les processus qui souhaitent attendre un sémaphore sont mis dans un état d'attente et placés dans la file d'attente pour les processus en attente. Un processus désirant un sémaphore binaire effectue une vérification d'état. Si le sémaphore est disponible, il change l'état du sémaphore et continue son exécution. Si le sémaphore est occupé, le noyau met le processus dans une file d'attente qui l'attend. L'état d'attente peut avoir un délai d'expiration, selon le support du noyau. Une fois que le processus occupant le sémaphore le libère, le noyau vérifie quel processus donnera accès la prochaine fois. La sélection peut être basée sur plusieurs options telles que:

- Processus hautement prioritaire
- Système de file d'attente FIFO.

Généralement. Il existe trois types de sémaphore:

- Sémaphores binaires (valeur de sémaphore de 0 ou 1 pour indiquer respectivement l'indisponibilité et la disponibilité)
- Comptage des sémaphores (valeur de sémaphore égale ou supérieure à 0 indiquant qu'il peut être acquis / libéré plusieurs fois)
- Sémaphores d'exclusion mutuelle (valeur de sémaphore de 0 ou 1 mais le nombre de verrous peut être égal ou supérieur à 0 pour le verrouillage récursif)

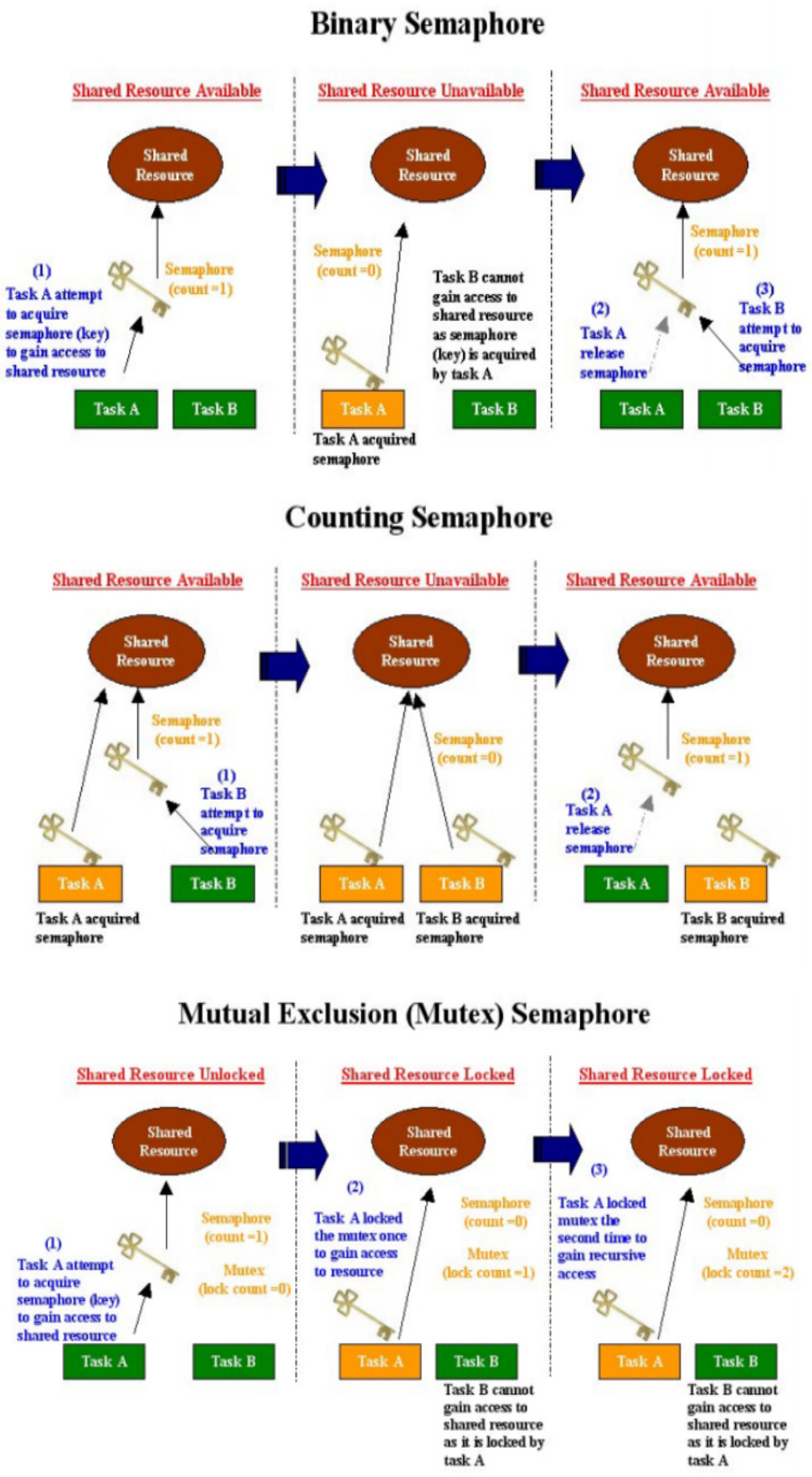


Figure .II. 9. Types de sémaphore [17].

- **Communication interprocessus**

La communication est un élément central de tout système d'exploitation. Les tâches coopérants (processus ou threads) communiquent et se synchronisent souvent. L'exécution d'une tâche particulière peut affecter une ou plusieurs autres tâches par communication. La communication entre les tâches implique le partage de données entre les tâches grâce au partage de l'espace mémoire, à la transmission de données, etc. Comme indiqué ci-dessous, il existe généralement plusieurs types de communication interprocessus (IPC) dans les systèmes d'exploitation en temps réel:

- **Structure de données partagée** : Le moyen le plus évident pour les tâches de communiquer est d'accéder à des structures de données partagées. Les instances de structures de données partagées comprennent des variables globales, des tampons linéaires, des tampons en anneau, des listes chaînées et des pointeurs, etc. La communication interprocessus utilisant une structure de données partagée est illustrée à **la figure.II. 11.**
- **File d'attente de messages :** Les files d'attente de messages permettent à un nombre variable de messages, chacun de longueur variable, d'être mis en file d'attente. Les tâches et les routines de service d'interruption (ISR) peuvent envoyer des messages à une file d'attente de messages et recevoir des messages d'une file d'attente de messages. La communication interprocessus à l'aide d'une file d'attente de messages est illustrée à **la figure .II.12.**
- **Signal :** Les signaux sont plus appropriés pour la gestion des erreurs et des exceptions que comme mécanisme de communication interprocessus à usage général. Toute tâche ou ISR peut générer un signal pour une tâche particulière. La tâche signalée suspend immédiatement son thread d'exécution actuel et exécute la routine de gestionnaire de signal spécifiée par la tâche lors de sa prochaine exécution. Le gestionnaire de signaux est appelé même si la tâche est bloquée.
- **Autres** :Certains autres mécanismes de communication peuvent exister dans différents systèmes d'exploitation en temps réel. Les canaux, qui sont des périphériques d'E / S virtuels, fournissent une interface alternative à la fonction de file d'attente de messages qui passe par le système d'E / S. Socket est un mécanisme de communication réseau interprocessus de base dans lequel les données sont envoyées d'une socket à une autre à travers le réseau. Les appels de procédure à distance (RPC) sont une fonction qui

permet à un processus sur une machine d'appeler une procédure exécutée par un autre processus sur la même machine ou sur une machine distante.

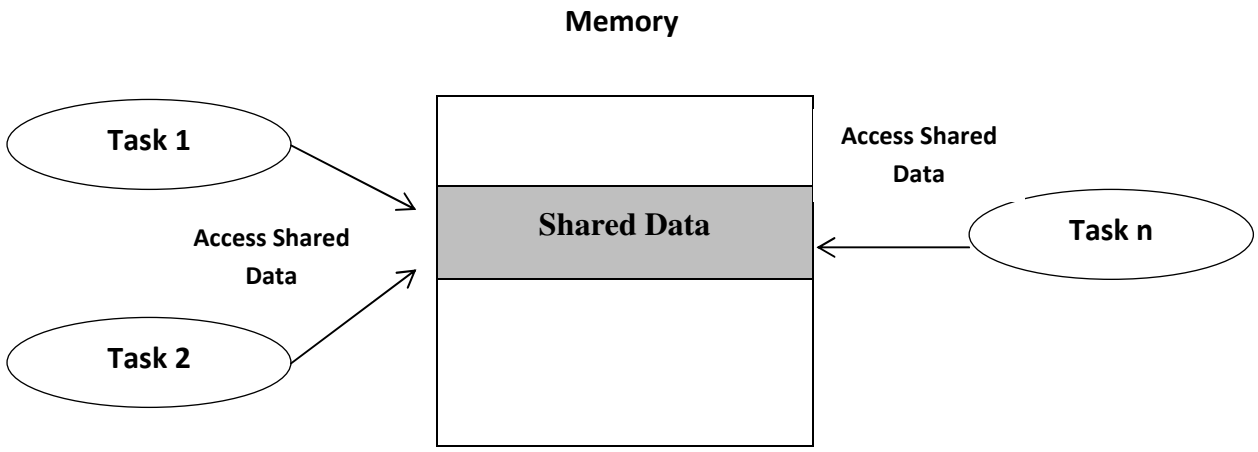


Figure .II. 10: Communication interprocessus via une structure de données partagée [17].

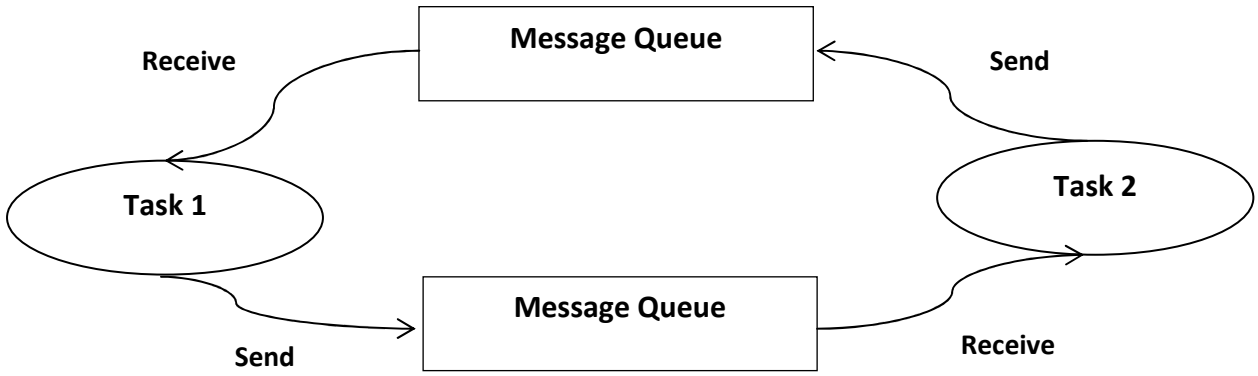


Figure .II. 11. Communication interprocessus via des files d'attente de messages [17].

II.3. 4 Gestion de la mémoire

Un RTOS intégré s'efforce généralement d'obtenir un faible encombrement en n'incluant que les fonctionnalités nécessaires aux applications de l'utilisateur. Il existe deux types de gestion de la mémoire dans les RTOS. Ce sont des gestions de pile et de tas.

Dans un RTOS multitâche, chaque tâche doit être allouée avec une quantité de mémoire pour stocker leurs contextes (c'est-à-dire des informations volatiles telles que le contenu des registres, le compteur de programmes, etc.) pour la commutation de contexte. Cette allocation de mémoire se fait en utilisant le modèle de bloc de contrôle de tâches (comme mentionné dans la section II.3.2). Cet ensemble de mémoire est

communément appelé pile du noyau et le processus de gestion est appelé Stack Management.

À la fin de l'initialisation d'un programme, la mémoire physique du MCU ou du MPU sera généralement occupée par le code de programme, les données de programme et la pile système. La mémoire physique restante est appelée tas. Cette mémoire de tas est généralement utilisée par le noyau pour l'allocation dynamique de mémoire d'espace de données pour les tâches. La mémoire est divisée en blocs de mémoire de taille fixe, qui peuvent être demandés par des tâches. Lorsqu'une tâche termine d'utiliser un bloc de mémoire, elle doit le renvoyer au pool. Ce processus de gestion de la mémoire du tas est appelé gestion du tas.

II.3. 5. Gestion de la minuterie

Dans les systèmes embarqués, les tâches système et utilisateur sont souvent planifiées pour s'exécuter après une durée spécifiée. Pour fournir une telle planification, il existe un besoin d'une interruption périodique pour garder une trace des retards et des délais. La plupart des RTOS offrent aujourd'hui à la fois des «minuteries relatives» qui fonctionnent en unités de graduation et des «minuteries absolues» qui fonctionnent avec la date et l'heure du calendrier. Pour chaque type de temporisateur, les RTOS fournissent un service de «délai de tâche», ainsi qu'un service «d'alerte de tâche» basé sur le mécanisme de signalisation (par exemple, des indicateurs d'événement). Un autre service de minuterie fourni consiste à respecter l'échéance des tâches en coopérant avec les planificateurs de tâches pour déterminer si les tâches ont respecté ou manqué leurs échéances en temps réel.

II.3.6. Gestion des interruptions et des événements

Une interruption est un mécanisme matériel utilisé pour informer la CPU qu'un événement asynchrone s'est produit. Un défi fondamental dans la conception RTOS est de prendre en charge les interruptions et de permettre ainsi un accès asynchrone aux structures de données RTOS internes. Le mécanisme de gestion des interruptions et des événements d'un RTOS fournit les fonctions suivantes:

- Définition du gestionnaire d'interruption
- Création et suppression d'ISR
- Référencement de l'état d'un ISR
- Activation et désactivation d'une interruption

- Modification et référencement d'un masque d'interruption et contribuent à assurer.
- Intégrité des données en empêchant les interruptions de se produire lors de la modification d'une structure de données
- Latences d'interruption minimales en raison de la désactivation des interruptions lorsque RTOS exécute des opérations critiques
- Réponses d'interruption les plus rapides qui ont marqué les performances préventives d'un RTOS
- Temps de fin d'interruption le plus court possible avec des frais généraux minimaux.

II.3.7. Gestion des E / S des appareils

Un noyau RTOS est souvent équipé d'un service de gestion des E / S de périphérique pour fournir un cadre uniforme (interface du programmeur d'application - «API») et une fonction de supervision pour un système embarqué pour organiser et accéder à un grand nombre de pilotes de périphériques matériels divers. Cependant, la plupart des API et superviseurs de pilotes de périphériques ne sont «standard» que dans un RTOS spécifique.

II.4 .Composants de RTOS [18]

- **Le planificateur (l'ordonnanceur)** : ce composant de RTOS indique que dans quel ordre, les tâches peuvent être exécutées, ce qui est généralement basé sur la priorité.

- **Multitraitement symétrique (SMP)** : c'est un certain nombre de tâches différentes qui peuvent être gérées par le RTOS afin que le traitement parallèle puisse être effectué.
- **Bibliothèque de fonctions** : C'est un élément important de RTOS qui agit comme une interface qui vous aide à connecter le noyau et le code d'application. Cette application vous permet d'envoyer les requêtes au noyau en utilisant une bibliothèque de fonctions afin que l'application puisse donner les résultats souhaités.
- **Gestion de la mémoire** : cet élément est nécessaire dans le système pour allouer de la mémoire à chaque programme, qui est l'élément le plus important du RTOS.

- **Latence de répartition rapide** : c'est un intervalle entre la fin de la tâche qui peut être identifiée par le système d'exploitation et le temps réel pris par le thread, qui est dans la file d'attente prête, qui a commencé le traitement.
- **Objets et classes de données définis par l'utilisateur** : le système RTOS utilise des langages de programmation comme C ou C ++, qui doivent être organisés en fonction de leur fonctionnement.

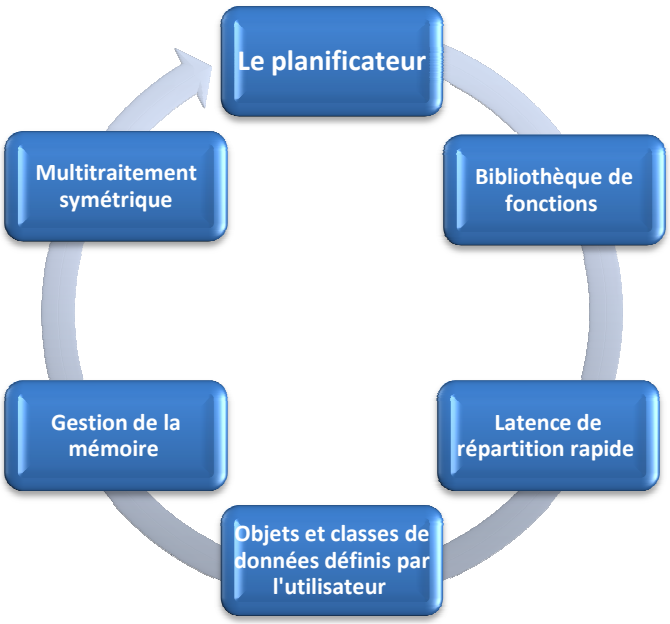


Figure .II. 12. Les composants de RTOS.

II.5 .Les RTOS actuels

Il existe de nos jours une grande variété de système d'exploitation temps réel, en comparaison avec les systèmes d'exploitation pour usage généraliste. La plupart d'entre eux fournissent les outils nécessaires pour le développement de systèmes temps réel destinés aux systèmes embarqués. On peut citer quelques-uns des plus utilise [19] : FreeRTOS, LynxOS, VxWorks, QNX , RT-Linux , et VRTX .

RTOS	Applications / Fonctionnalités
FreeRTOS	- Les plus utilisés dans le marché des systèmes d'exploitation temps réel. -Faible empreinte, portable, préemptif et Open source pour les microcontrôleurs - Porté sur 33 architectures différentes.

	- Un nombre illimité de tâches peut être exécuté simultanément et sans contrainte.
LynxOS	-Applications complexes et difficiles en temps réel Système d'exploitation multiprocessus et multithread compatible POSIX - Prise en charge par les architectures x86, ARM, PowerPC
VxWorks	-RTOS le plus largement adopté dans l'industrie embarquée. -Utilisé dans les célèbres robots rover de la NASA Spirit and Opportunity. -Certifié par plusieurs agences et normes internationales pour les systèmes temps réel, la fiabilité et les applications critiques pour la sécurité.
QNX	-Les RTOS les plus traditionnels du marché. -Architecture micro-noyau; totalement compatible avec le POSIX -Certifié par les normes FADO-278 et MIL-STD-1553.
RT-Linux	-Noyau temps réel dur. -Bonnes performances en temps réel, mais pas de certification.
VRTX	-Convient aux systèmes embarqués traditionnels basés sur des cartes et aux architectures SoC. -Prise en charge par ARM, MIPS, PowerPC et autres architectures RISC.

Tableau. II.1. Quelques exemples des RTOS le plus utilisé.

II.6 .Facteurs de sélection d'un RTOS

Voici les facteurs essentiels que vous devez prendre en compte pour sélectionner RTOS [18]:

- **Performances** : les performances sont le facteur le plus important à prendre en compte lors de la sélection d'un RTOS**.**
- **Middleware** : s'il n'y a pas de support middleware dans le système d'exploitation en temps réel, le problème de l'intégration des processus prend du temps.
- **Sans erreur** : les systèmes RTOS sont sans erreur. Par conséquent, il n'y a aucune chance d'obtenir une erreur lors de l'exécution de la tâche.
- **Utilisation du système intégré** : les programmes de RTOS sont de petite taille. Nous utilisons donc largement RTOS pour les systèmes embarqués.

- **Consommation maximale** : nous pouvons atteindre une consommation maximale avec l'aide de RTOS.
- **Transfert de tâches:** le temps de **transfert** des tâches est très inférieur.
- **Caractéristiques uniques** : Un bon RTS devrait être capable, et il a quelques fonctionnalités supplémentaires comme la façon dont il fonctionne pour exécuter une commande, une protection efficace de la mémoire du système, etc.
- **Performance 24/7** : RTOS est idéal pour les applications qui doivent fonctionner 24/7.

II.7.Conclusion

RTOS est un système d'exploitation destiné à servir des applications en temps réel qui traitent les données au fur et à mesure qu'elles arrivent, il occupe très moins de mémoire et consomme moins de ressources.

Dans ce chapitre nous avons présenté les systèmes d'exploitation en temps réel. Dans la première partie, on a défini les systèmes d'exploitation en temps réel avec leurs catégories tels que temps réel dur, temps ferme et temps réel souple. Puis on a donné l'architecteur d'un RTOS et les trois grandes catégories de modèles de noyaux disponibles et leurs six services communs tels que gestion des tâches, synchronisation des tâches et les communications entre les tâches, gestion de la mémoire, gestion de la minuterie, gestion des interruptions et des événements, et gestion des E / S des appareils. On a fini par certain nombre de RTOS disponibles chacun avec ses fonctionnalités distinctes et ciblés pour un ensemble spécifique d'applications et les facteurs essentiels que vous devez prendre en compte pour sélectionner RTOS.

Chapitre **III** *:*

FreeRTOS

III. 1. Introduction

Le planificateur dans un système d'exploitation en temps réel (RTOS) est conçu pour fournir un modèle d'exécution prévisible (normalement décrit comme *déterministe)*. Ceci est particulièrement intéressant pour les systèmes embarqués, comme les appareils Arduino, car les systèmes embarqués ont souvent des exigences en temps réel.
Les planificateurs traditionnels en temps réel, tels que le planificateur utilisé dans FreeRTOS atteignent le déterminisme en permettant à l'utilisateur d'attribuer une *priorité* à chaque thread d'exécution [2]. Le planificateur utilise ensuite la priorité pour savoir quel thread d'exécution exécuter ensuite. Dans FreeRTOS, un thread d'exécution est appelé une *tâche*. D'une part free RTOS [20] a développé en partenariat avec les plus grands fabricants de puces au monde sur une période de 15ans, et maintenant téléchargé toutes les 175 secondes, Free RTOS est un system d'exploitation en temps réel (RTOS) leader du marché pour les microcontrôleurs et les petits microprocesseurs. Free RTOS est uniquement détenu, développé et maintenu par Real Time Engineers Ltd.

Dans ce chapitre on s'intéressera à la présentation de FreeRTOS, la définition, pourquoi est- il considéré comme un bon choix, et l'utilisation avec la carte Arduino.

III. 2. Définition de FreeRTOS

FreeRTOS est une classe de RTOS qui est conçue pour être suffisamment petite pour fonctionner sur un microcontrôleur - bien que son utilisation ne se limite pas aux applications de microcontrôleur [20].

FreeRTOS peut être construit avec environ vingt compilateurs différents, et peut fonctionner sur plus de trente architectures de processeurs différentes. Chaque combinaison prise en charge de compilateur et de processeur est considérée comme un port FreeRTOS distinct [21].

Un microcontrôleur est un petit processeur aux ressources limitées qui incorpore, sur une seule puce, le processeur lui-même, une mémoire morte (ROM ou Flash) pour contenir le programme à exécuter et la mémoire vive (RAM) nécessaire aux programmes qu'il exécute. En général, le programme est exécuté directement à partir de la mémoire morte [20].

Les microcontrôleurs sont utilisés dans des applications profondément intégrées (ces applications où vous ne voyez jamais réellement les processeurs eux-mêmes, ni les logiciels

qu'ils exécutent) qui ont normalement un travail très spécifique et dédié à faire. Les contraintes de taille et la nature de l'application finale dédiée justifient rarement l'utilisation d'une implémentation RTOS complète - ou rendent en fait l'utilisation d'une implémentation RTOS complète possible. FreeRTOS fournit donc la fonctionnalité de planification en temps réel de base, la communication inter-tâches, la synchronisation et les primitives de synchronisation uniquement. Cela signifie qu'il est plus précisément décrit comme un noyau en temps réel ou un exécutif en temps réel. Des fonctionnalités supplémentaires, telles qu'une interface de console de commande ou des piles réseau, peuvent alors être incluses avec des composants complémentaires.

Voici quelques raisons pour lesquelles FreeRTOS est un bon choix pour votre prochaine application – FreeRTOS [20]:

- Fournit une solution unique et indépendante pour de nombreuses architectures et outils de développement différents.
- Est connu pour être fiable. La confiance est assurée par les activités entreprises par le projet frère SafeRTOS.
- Est riche en fonctionnalités et en cours de développement actif continu.
- A un minimum de ROM, de RAM et de traitement. En règle générale, une image binaire du noyau RTOS sera dans la région de 6K 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 concernent uniquement les nombreuses applications de démonstration.
- Est vraiment gratuit pour une utilisation dans des applications commerciales.

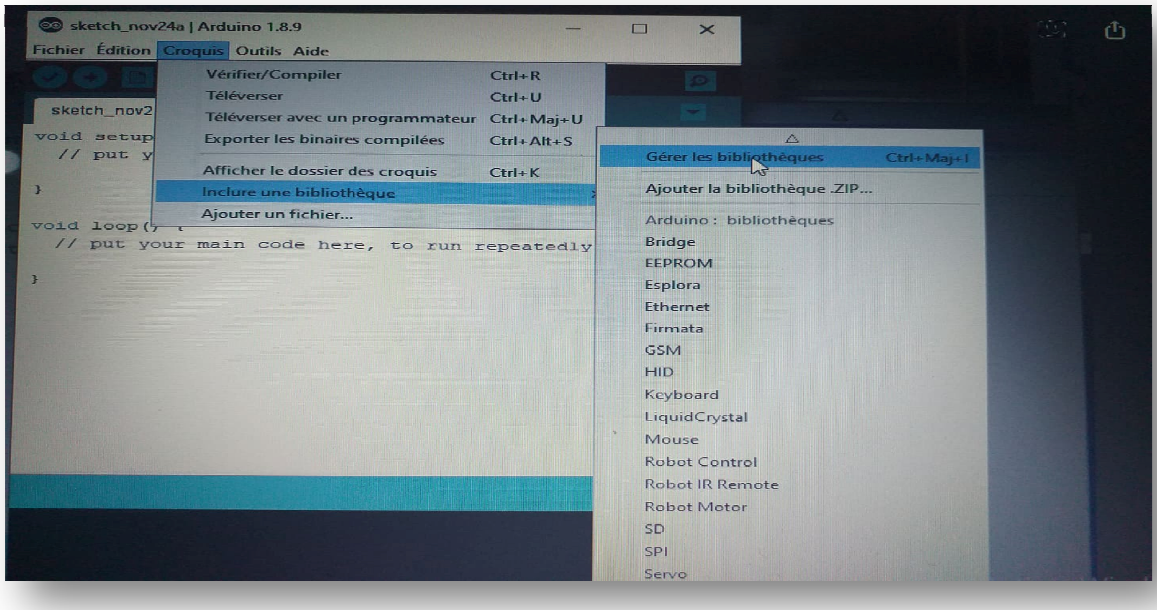
III. 3. Politique de Scheduling de FreeRTOS

Le noyau FreeRTOS prend en charge deux types de politique de Scheduling [21]:

- Time Slicing Scheduling Policy (Politique de planification par tranches de temps) : Ceci est également connu sous le nom d'algorithme à tour de rôle. Dans cet algorithme, toutes les tâches de priorité égale obtiennent le processeur en parts égales de temps processeur.
- Fixed Priority Preemptive Scheduling (Planification préventive à priorité fixe): cet algorithme de planification sélectionne les tâches en fonction de leur priorité. En d'autres termes, une tâche à haute priorité obtient toujours le processeur avant une tâche à faible priorité. Une tâche de faible priorité s'exécute uniquement lorsqu'il n'y a pas de tâche de haute priorité à l'état prêt.

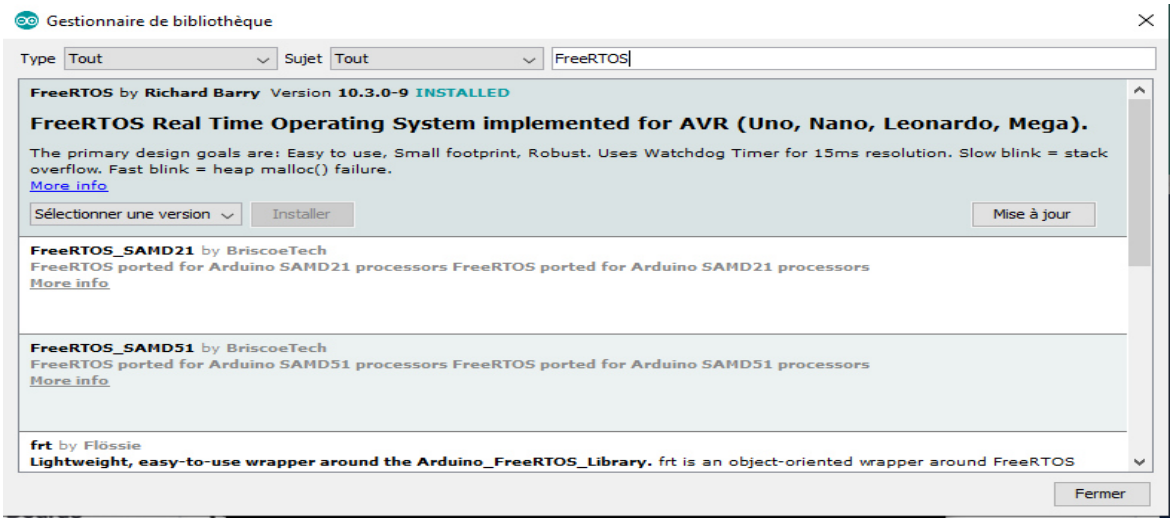
III.4. Utilisation de FreeRTOS avec Arduino

- **Téléchargez et installez FreeRTOS dans Arduino IDE** : La bibliothèque FreeRTOS peut également être installée directement via le gestionnaire de bibliothèque Arduino. Pour cela, ouvrez Arduino IDE et allez dans **Croquis Inclure la bibliothèque** et cliquez sur **Gérer les bibliothèques**.



III. 1. Chemin pour ouvrir le gestionnaire de librairies.

Après cela, tapez **FreeRTOS** dans la fenêtre **de recherche**, cette bibliothèque apparaîtra. Après cela, cliquez sur le bouton d'installation



III. 2.L'étape de télécharger bibliothèque FreeRTOS.

III. 5. Gestion des tâches [22,24]

- **Création des tâches**

```
BaseType_t xTaskCreate (TaskFunction_t pvTaskCode, const char * const pcName
configSTACK_DEPTH_TYPE usStackDepth,   void * pvParameters, UBaseType_t     uxPriority,
TaskHandle_t * pxCreatedTask   );
```

Créez une nouvelle tâche et ajoutez-la à la liste des tâches prêtes à être exécutées

Paramètres:

pvTaskCode	Pointeur vers la fonction d'entrée de tâche (juste le nom de la fonction qui implémente la tâche). Les tâches sont normalement implémentées comme une boucle infinie et ne doivent jamais tenter de retourner ou de quitter leur fonction d'implémentation. Les tâches peuvent cependant se supprimer elles-mêmes .
pcName	Un nom descriptif pour la tâche. Ceci est principalement utilisé pour faciliter le débogage, mais peut également être utilisé pour obtenir un descripteur de tâche . La longueur maximale du nom d'une tâche est définie à l'aide du paramètre configMAX_TASK_NAME_LEN dans FreeRTOSConfig.h .
usStackDepth	Le nombre de mots (et non d'octets!) À allouer pour être utilisés comme pile de la tâche. Par exemple, si la pile a une largeur de 16 bits et usStackDepth est de 100, alors 200 octets seront alloués pour être utilisés comme pile de la tâche. Comme autre exemple, si la pile a une largeur de 32 bits et que usStackDepth est de 400, 1600 octets seront alloués pour être utilisés comme pile de la tâche. La profondeur de pile multipliée par la largeur de pile ne doit pas dépasser la valeur maximale pouvant être contenue dans une variable de type *size_t*.
pvParamètres	Une valeur qui sera transmise à la tâche créée en tant que paramètre de la tâche. Si *pvParameters* est défini sur l'adresse d'une variable, la variable

	doit toujours exister lorsque la tâche créée s'exécute , il n'est donc pas valide de transmettre l'adresse d'une variable de pile.
uxPriority	La priorité à laquelle la tâche créée sera exécutée. Les systèmes prenant en charge MPU peuvent éventuellement créer une tâche en mode privilégié (système) en définissant le bit portPRIVILEGE_BIT dans uxPrriority. Par exemple, pour créer une tâche privilégiée à la priorité 2, définissez uxPriority sur (2 | *portPRIVILEGE_BIT*).
pxCreatedTask	Utilisé pour passer un handle à la tâche créée en dehors de la fonction *xTaskCreate (). pxCreatedTask* est facultatif et peut être défini sur **NULL**.
Returned value	Si la tâche a été créée avec succès, *pdPASS* est renvoyé. Sinon, *errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY* est renvoyé.

- **Suppression des tâches**

```
void vTaskDelete(TaskHandle_t pxTaskToDelete);
```

Paramètres:

La fonction API **vTaskDelete** () FreeRTOS est utilisée pour supprimer des tâches. Avec l'aide de cette fonction API, toute tâche peut se supprimer elle-même et peut également supprimer d'autres tâches en passant une référence par un gestionnaire à la fonction **vTaskDelete ().**

PxTaskToDelete : Le descripteur de la tâche à supprimer (la tâche objet). Une tâche peut se supprimer elle-même en passant NULL à la place d'un descripteur de tâche valide.

- **Contrôle des tâches**
  - **Blocage d'une tâche**

```
void vTaskDelay (const TickType_t xTicksToDelay);
```

Retardez une tâche pour un nombre donné de ticks. Le temps réel pendant lequel la tâche reste bloquée dépend du taux de ticks. La constante *portTICK_PERIOD_MS* peut être utilisée pour calculer en temps réel à partir du taux de ticks - avec la résolution d'une période de ticks.

Paramètres:

xTicksToDelay Durée, en périodes de graduation, que la tâche appelante doit bloquer.

- **Modification de la priorité d'une tâche**

void vTaskPrioritySet (TickHandle_t pxTask, UBaseType_t uxNewPriority);

Paramètres:

pxTask — Le descripteur de la tâche dont la priorité est en cours de modification (la tâche objet). Consultez le paramètre pxCreatedTask de la fonction API xTaskCreate () pour obtenir des informations sur l'obtention de descripteurs de tâches.
Une tâche peut modifier sa propre priorité en passant NULL à la place d'un descripteur de tâche valide.

uxNewPriority — La priorité à laquelle la tâche objet doit être définie.

- **Contrôle du noyau**

void vTaskStartScheduler(void) ;

Démarre le processus du noyau temps réel.

- **Synchronisation des tâches FreeRTOS**

- **Création de Sémaphore binaire**

SemaphoreHandle_t xSemaphoreCreateBinary (void);

Les sémaphores binaires sont le moyen le plus simple de synchroniser les tâches.

- **xSemaphoreTake**

xSemaphoreTake(SemaphoreHandle_t xSemaphore, xTicksToWait);

Primitive de demande de ressource. Cette opération est équivalente à une opération P (). Si la ressource n'est pas disponible, la tâche sera bloqué jusqu'à la disponibilité de la ressource ou l'écoulement du délai.

Paramètres:

xSemaphore l'identificateur de la sémaphore (handle)

xTicksToWait : temps d'attente de la disponibilité de la ressource. La valeur portMAX_DELAY permet d'avoir une attente infinie.

- **xSemaphoreGive**

```
xSemaphoreGive( SemaphoreHandle_t xSemaphore );
```

Primitive de libération de ressource. Cette opération est équivalente à une opération V ()

- **Création de compte Sémaphore**

```
SemaphoreHandle_t xSemaphoreCreateCounting( UBaseType_t uxMaxCount,
UBaseType_t uxInitialCount );
```

Paramètres:

uxMaxCount La valeur de comptage maximale pouvant être atteinte. Lorsque le sémaphore atteint cette valeur, il ne peut plus être «donné».

uxInitialCount La valeur de comptage attribuée au sémaphore lors de sa création.

- **Création de Mutex**

```
SemaphoreHandle_t xSemaphoreCreateMutex ( void );
```

- **Suppression de Sémaphore**

```
void vSemaphoreDelete ( SemaphoreHandle_t xSemaphore );
```

Supprime un sémaphore précédemment créé à l'aide d'un appel à *vSemaphoreCreateBinary (), xSemaphoreCreateCounting (), xSemaphoreCreateMutex ().*

Paramètres:

xSémaphore Le handle du sémaphore en cours de suppression.

III. 6. Conclusion

Dans ce chapitre, et dans un premier temps, nous avons présenté une classe de RTOS qui est le FreeRTOS (les raisons pour lesquelles est un bon choix, politique de scheduling). Dans la deuxième partie de ce chapitre nous avons donné l'utilisation de FreeRTOS avec Arduino et les fonctions tels que la création des tâches, la suppression, la synchronisation des tâches.

Chapitre IV

Application et réalisation

IV.1. Introduction :

Dans ce chapitre, nous allons évoqué l'application d'un système d'exploitation en temps réel (RTOS) qui est **FreeRTOS** sur la plateforme Arduino au sein d'une maison intelligente. Ainsi que la programmation, la simulation et réalisation.

IV.2. Explication de projet

L'intérêt de ce travail était de l'importance de l'utilisation de RTOS dans les systèmes en temps réel. Ainsi notre dévolu est tombé sur le prototype d'une maison intelligente .Dans l'objectif est de localisé une réponse de certaine exigence de plusieurs fonctions dans un délai court est strictement défini. Pour cela nous avons testé deux programmes sur la plateforme Arduino : Programme multitâche « Arduino simple » et programme multitâche dans une classe de RTOS qui est le FreeRTOS.

.

Figure. IV. 2. Photo de la maison intelligent

IV.3. Partie hard

Dans l'objectif de réaliser ce travail une multitude de composant électronique à était couplé à une carte ou nous avons pu décliné les différents gammas de la carte ARDUINO sous ce déférentes modules.

IV.3. 1. Présentation de la carte électronique

Arduino est une plate-forme de prototypage d'objets interactifs à usage créatif constituée d'une carte électronique et d'un environnement de programmation. Sans tout connaitre ni tout comprendre de l'électronique, cet environnement matériel et logiciel permet a l'utilisateur de

formuler ses projets par l'expérimentation directe avec l'aide de nombreuses ressources disponibles en ligne.

Pont tendu entre le monde réel et le monde numérique, Arduino permet d'étendre les capacités des relations humain/machine ou environnement/machine. Arduino est un projet en source ouverte : La communauté importante d'utilisateurs et de concepteurs permet à chacun de trouver les réponses à ses questions [25].

IV.3.1.1. Arduino Mega

Arduino Mega 2560 est une carte microcontrôleur basée sur l'ATmega2560 (fiche technique). Il dispose de 54 broches d'entrée / sortie numériques (dont 14 peuvent être utilisées comme sorties PWM), 16 analogiques entrées, 4 UART (ports série matériels), un oscillateur à quartz 16 MHz, une connexion USB, un prise d'alimentation, un en-tête ICSP et un bouton de réinitialisation. Il contient tout le nécessaire pour soutenir le microcontrôleur; connectez-le simplement à un ordinateur avec un câble USB ou alimentez-le avec un ACto-DC adaptateur ou batterie pour commencer. Le Mega est compatible avec la plupart des boucliers conçus pour l'Arduino Duemilanove ou Diecimila. L'Arduino Mega peut être alimenté via la connexion USB ou avec une alimentation externe, La carte peut fonctionner sur une alimentation externe de 6 à 20 volts. Si fourni avec moins de 7V, cependant, la broche 5V peut fournir moins de cinq volts et la carte peut être instable. Si vous utilisez plus de 12 V, le régulateur de tension peut surchauffer et endommager la carte.

La Mega2560 diffère de toutes les cartes en ce qu'elle n'utilise pas la puce de pilote FTDI USB-série. Au lieu de cela, il intègre l'Atmega8U2 programmé comme un convertisseur USB-série. Les broches d'alimentation sont les suivantes [26]:

• **VIN :** La tension d'entrée de la carte Arduino lorsqu'elle utilise une source d'alimentation externe (par opposition à 5 volts depuis la connexion USB ou une autre source d'alimentation régulée). Vous pouvez fournir une tension via cette broche, ou, si fournir une tension via la prise d'alimentation, y accéder via cette broche.

• **5V :** L'alimentation régulée utilisée pour alimenter le microcontrôleur et les autres composants de la carte. Ce peut provenir soit du VIN via un régulateur embarqué, soit être alimenté par USB ou une autre alimentation régulée 5V.

• **3V3 :** Une alimentation de 3,3 volts générée par le régulateur de bord. La consommation maximale de courant est de 50 mA.

• **GND :** Broches de masse.

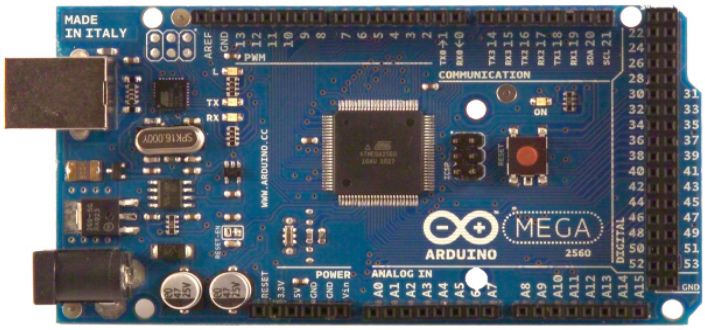


Figure. IV. 3. Photo réel de l'Arduino Mega.

IV.3. 1.2. L'intérêt d'utilisation Arduino méga

Pour notre projet ambitieux, on se tourne vers la carte Arduino Méga vu le volume et le nombre de processus utilisés dans cette application qui sera présenté dans ce chapitre. L'intérêt principal de cette carte est de faciliter la mise en œuvre d'une telle commande qui sera détaillée par la suite [26].

IV.3.2. Partie matérielles

- **Bipeur :** (en anglais Buzzer) est un élément électromécanique ou piézoélectrique qui produit un son caractéristique quand on lui applique une tension : le bip. Certains nécessitent une tension continue, d'autres nécessitent une tension alternative [27].
- **MQ-2 Capteur de Gaz/Fumée :** est un semi-conducteur capteur de gaz/fumée qui détecte la présence du gaz/fumée à des concentrations de 300 ppm à 10000 ppm. Le simple interface de tension analogique du capteur ne nécessite qu'une seule broche d'entrée analogique de votre microcontrôleur.

  Le capteur de gaz méthane MQ-2 détecte la concentration de gaz/fumée dans l'air et sorties le résultat comme une tension analogique. La concentration de détection gamme de 300 ppm à 10000 ppm est appropriée pour la détection des fuites. Le capteur peut fonctionner à des températures allant de -10 à 50 ° C et consomme moins de 150 mA à 5 V [28].
- **LDR (Light Dépendent Résistor) :** Une photorésistance, on la désigne aussi par LDR, est un composant dont la valeur en ohms dépend de la lumière à la quelle il est exposé [29].
- **Servomoteur :** Un servomoteur (souvent abrégé en « Servo », provenant du latin *servus* qui signifie « esclave ») est un moteur capable de maintenir une opposition à un

effort statique et dont la position est vérifiée en continu et corrigée en fonction de la mesure. C'est donc un système asservi.

On a utilisé un servomoteur, communément appelé "servomoteur 9 grammes", par souci de consommation électrique et la plupart des servomoteurs fonctionnent en 5 volts [30].

- **LCD (Liquid Cristal Display) :** Les afficheurs LCD 16*2 (16 caractères et 2 lignes) sont des modules compacts intelligents et nécessitent peu de composants externes pour un bon fonctionnement. Ils consomment relativement peu (de 1 à 5 mA), dotés d'un rétro-éclairage de l'affichage. Cette fonction fait appel à des LED montées derrière l'écran du module, cependant, cet éclairage est gourmand en intensité (de 80 à 250 mA).Aujourd'hui ces écrans sont utilisés dans presque tous les affichages électroniques [31].
- **LED (Light Emitting Diode) :** Diode électroluminescente, on la désigne aussi par LED est un dispositif optoélectronique capable d'émettre de la lumière lorsqu'il est parcouru par un courant électrique [32].
- **Résistance :** Résistance à film métallique est un composant électronique ou électrique dont la principale caractéristique est d'opposer une plus ou moins grande résistance (mesurée en ohms) à la circulation du courant électrique [33].
- **Relais :** Un relais est un appareil dans lequel un phénomène électrique (courant ou tension) contrôle la commutation On / Off d'un élément mécanique ou d'un élément électronique. C'est en quelque sorte un interrupteur que l'on peut actionner à distance, et où la fonction de coupure est dissociée de la fonction de commande [34].
- **Clavier :** Le clavier est utilisé comme périphérique d'entrée pour lire la touche pressée par l'utilisateur et pour la traiter. Le clavier 4x4 se compose de 4 lignes et 4 colonnes. Les commutateurs sont placés entre les lignes et les colonnes. Un appui sur une touche établit une connexion entre la ligne et la colonne correspondantes, entre lesquelles le commutateur est placé [35].

IV.4. Partie Soft

Nous avons utilisé deux logiciels pour réaliser ce travail, tels que Arduino IDE, et ISIS.

IV.4.1. Le logiciel Arduino

Le logiciel de programmation de la carte Arduino de code (langage proche de C). Une fois, le programme tape ou modifié au clavier, il sera transféré et mémorisé dans la carte à travers de la liaison USB. Le câble USB alimente à la fois en énergie la carte et transporte aussi l'information, ce programme appelé IDE Arduino (l'Arduino IDE est le plus utilise, ainsi, il est disponible sur le site officiel pour télécharger gratuitement (www.arduino.cc)).

Comme n'importe quel langage de programmation, une interface souple et simple est exécutable sur n'importe quel système d'exploitation, Arduino base sur la programmation en **C** [25].

IV.4.1.1. Présentation de « IDE »

L'interface du logiciel Arduino se présente de la façon suivante :

On a numérote les informations les plus importantes de 1 a 8, on peut distinguer trois partis, 1-3 le logiciel, la carte, le nom. Ensuite le 4-5 sont des bottons permettant de vérifier et téléverser le code. Pour finir de 6-7 c'est le code :

1. La version du logiciel de l'Arduino, il faut faire attention si nous essayons un code sur internet il n'est pas forcement compatible si il n'est pas dans la même version.
2. Le nom du sketch qui correspond _a la date du jour ou nous avons ouvert notre sketch, il suit de l'enregistrer pour qu'il change de nom.
3. Le modèle de l'Arduino sur le port qui correspond.
4. Le bouton valider  permet de vérifier le code pour voir s'il y a des erreurs dedans.
5. Le bouton téléverser  permet de transmettre le code sur l'Arduino une fois fini. Il est conseille d'enregistrer et de vérifier le code avant de cliquer sur ce bouton.
6. void setup () :C'est la partie du code qui va s'exécuter lors du branchement de la carte Arduino ou lorsqu'on clique sur le bouton rouge reset. Elle permet d'initialiser les variables, le sens des broches et les librairies utilisées. Même si nous mettons rien dedans il le laisser dans le code.
7. void loop () : Est une fonction qui va s'exécuter sous forme de boucle (loop en Anglais), qui permet au programme de s'exécuter et de répondre. Cette fonction est aussi obligatoire dans le code.
8. Le moniteur série affiche des informations de base en temps réel. Il peut par exemple remplacer un écran LCD, parce qu'il est directement sur l'ordinateur.

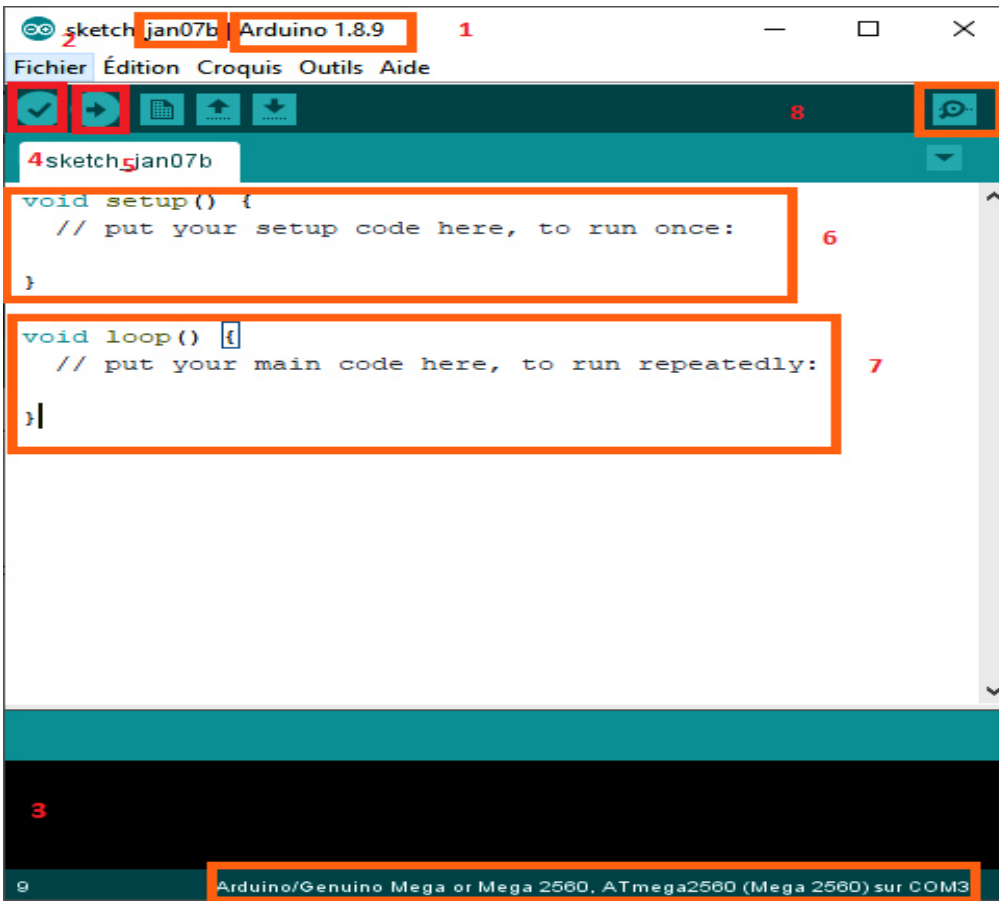


Figure. IV. 4. Interface de logiciel Arduino.

IV.4.2. ISIS

Le logiciel ISIS de Proteus Professional est principalement connu pour éditer des schémas électriques. Par ailleurs, le logiciel permet également de simuler ces schémas ce qui permet de déceler certaines erreurs dès l'étape de conception. Indirectement, les circuits électriques conçus grâce à ce logiciel peuvent être utilisé dans des documentations car le logiciel permet de contrôler la majorité de l'aspect graphique des circuits [36].

IV.4.2.1. Simulation de l'application sur ISIS

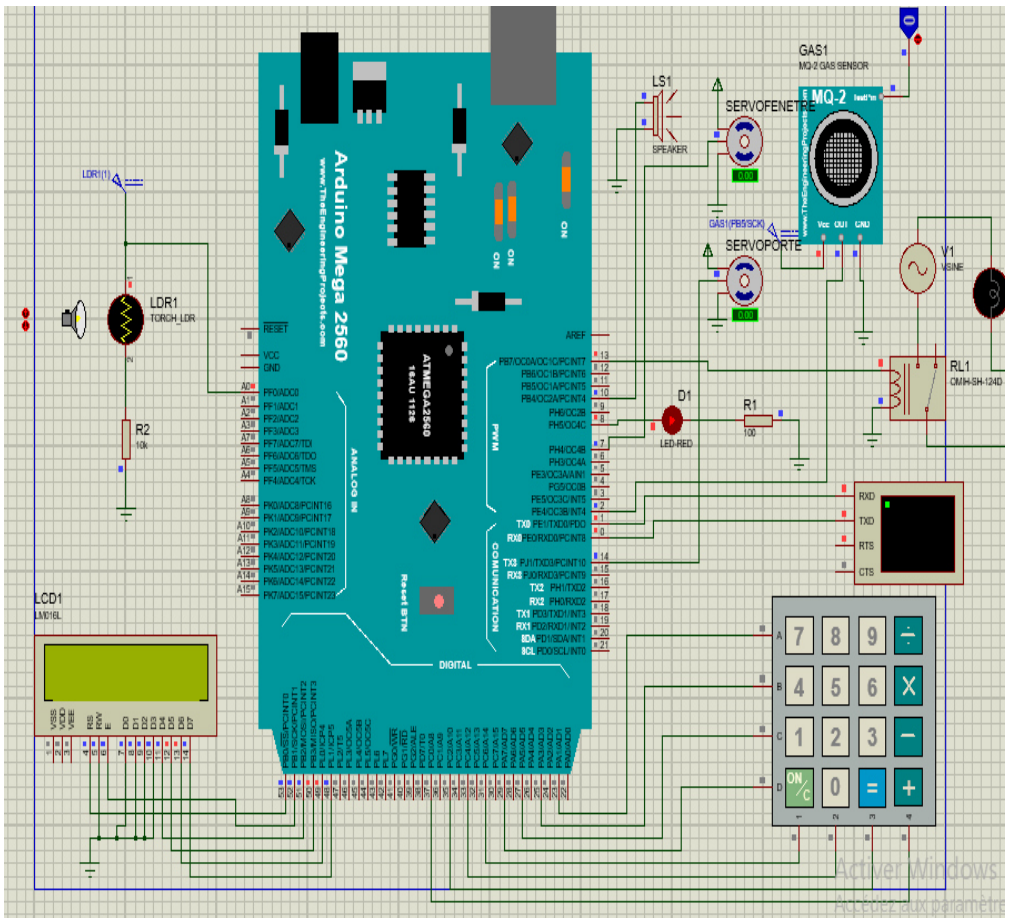


Figure. IV. 5. Simulation sur ISIS.

IV.5. L'organigramme de fonctionnement d'application avec multitâche simple Arduino

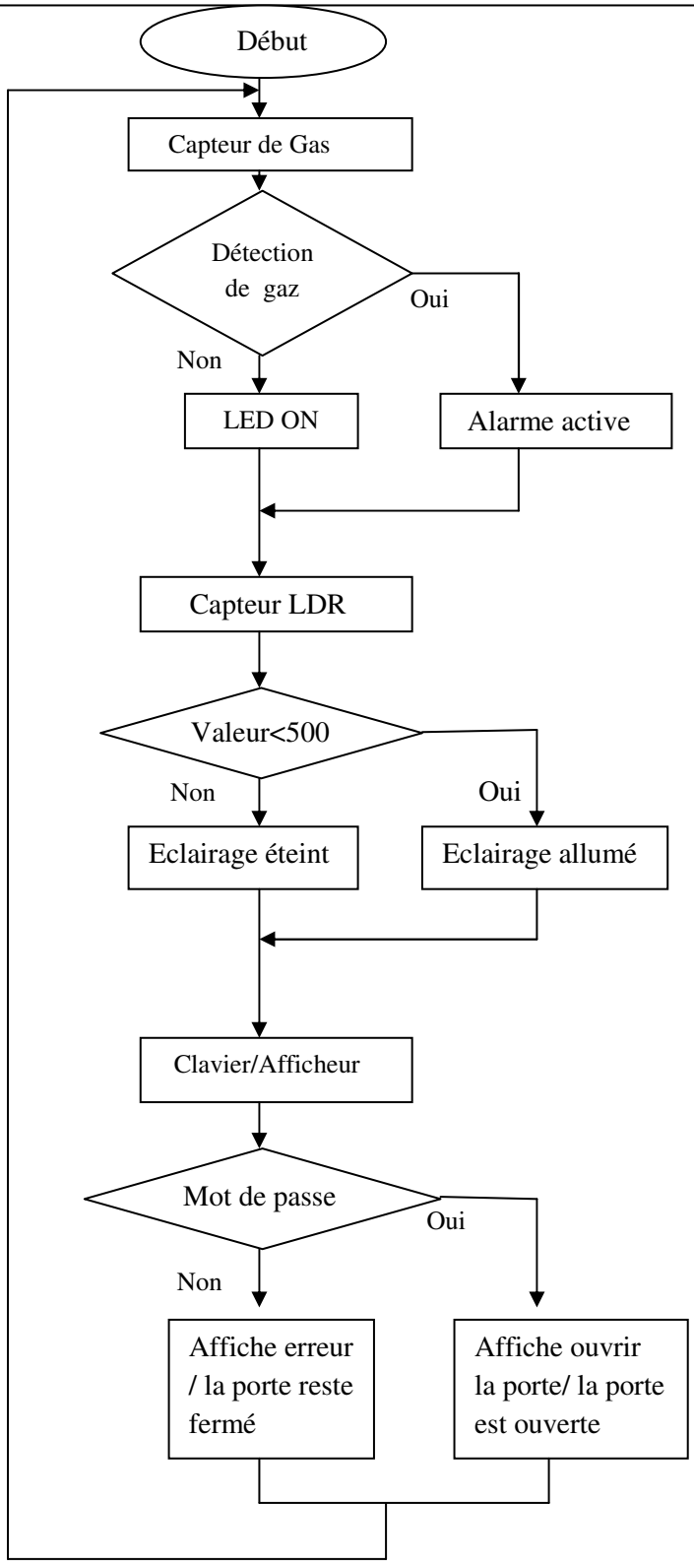


Figure. IV. 6.L'organigramme de fonctionnement d'application avec Multitâche simple Arduino.

IV.5.1.Discussion

Comme ca était précédemment mentionné dans l'organigramme de la maison intelligente commandé par le programme multifonction simple Arduino Mega.

Ce système en question consiste à placer un capteur de gaz MQ2 à fin de détecter la présence de gaz (monoxyde, butane, méthane) avec une concentration qui dépasse le seuil de détection du capteur. La Led reste allumée durant l'absence de fuit de gaz.

Dans le cas de la présence du gaz, le système déclenche l'alarme et envoie un signal pour exécuté des multifonctions en Arduino comme est énuméré le clignotement de la Led, le bip sonore et la rotation du servomoteur à un angle qui permet d'ouvrir la fenêtre de la maison automatiquement.

A l'aide d'un relais et d'une photorésistance, nous avons contrôlé l'éclairage de la maison, l'éclairage s'éteint lors de la présence de la lumière (valeur de LDR supérieur 500 ohm) et s'allume dans le cas contraire.

Dans la dernière partie de l'organigramme nous avons déjà programmé le fonctionnement du clavier et de l'afficheur LCD, Car ce qui nous intéresse, sont les caractères qui s'affichent sur LCD lors de la pression sur la touche correspondante du clavier. Ainsi pour l'ouverture la porte automatiquement nécessité d'introduire un mot de passe est impérative .Dans ce cas précis la afficheur déligne « ouvrez la porte » le servomoteur tourne à un angle pour ouvrir la porte automatiquement .En cas de code erroné le tableau affiche « erreur » dans ce cas , comme c'est rien n'était à faire .

IV.6. Conception des communications et synchronisations des tâches avec FreeRTOS

La première phase consiste a la mise en ouvre est l'introduction de la bibliothèque FreeRTOS dans IDE , puis de déclarée les fonctions (la fonction dont quelle traitement se trouve dans une boucle infinie) , et enfin crée les tâches (on a déterminé leurs tailles, priorité et paramètres) ainsi que leur regroupement si possible. En plus de crée 8 tâches correspondant aux 8 processus fonctionnels.

Nous allons regrouper les fonctions « clignotement du LED», « Rotation du servomoteur» et du «bip Sonors » dans la tache « état d'alarme » du fait qu'elles doivent être déclenchées à la même période. Tandis que les autres tâches « ouverture de la porte » et

« tâche commande» seront nécessairement séparées pour qu'on puisse les manipuler séparément, car elles sont propres à chaque processus.

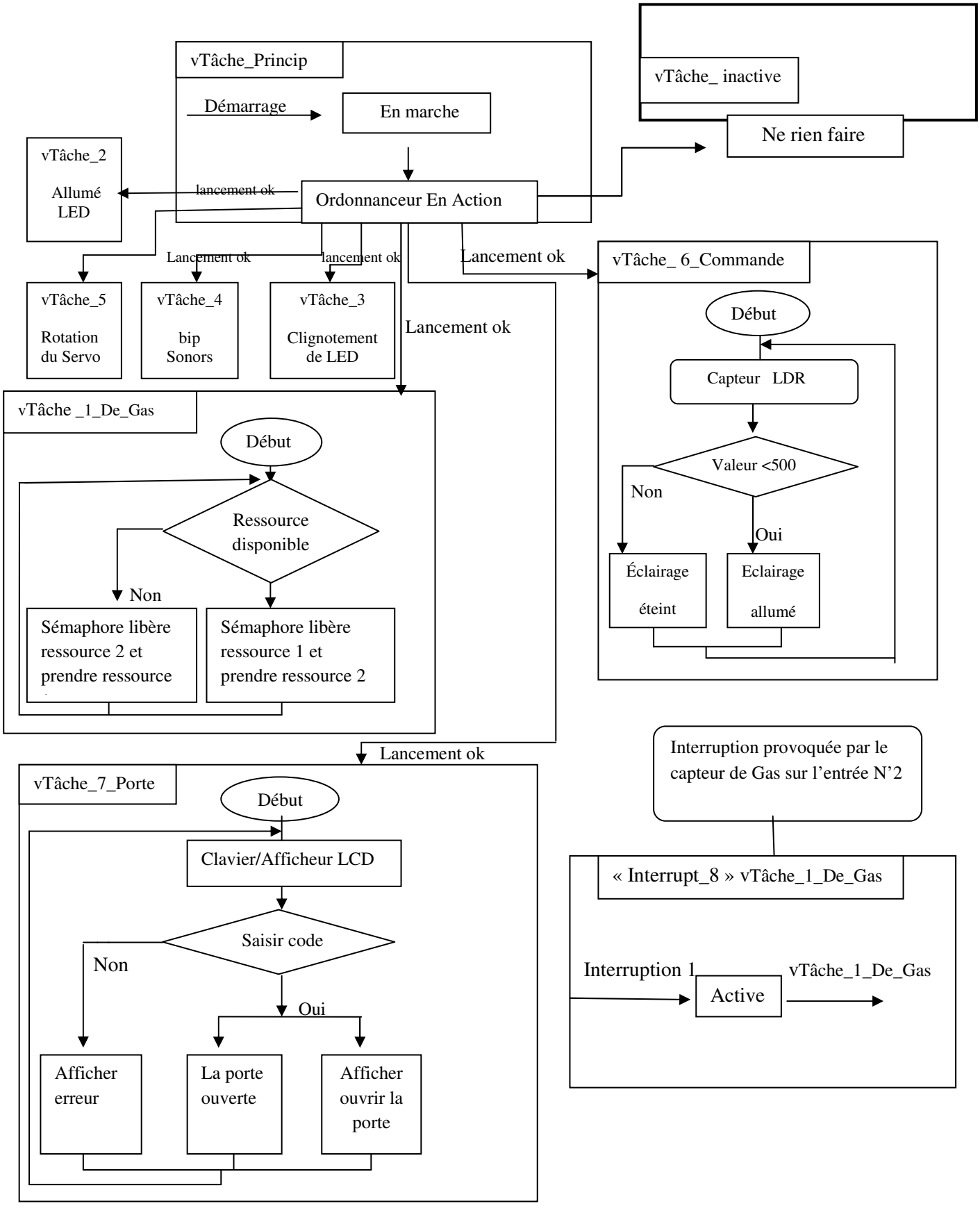


Figure. IV. 7. L'organigramme de fonctionnement d'application avec FreeRTOS.

IV.6.1.Discussion

- Un algorithme a été développé dans la plate-forme Arduino IDE en utilisant les bibliothèques FreeRTOS. L'ordonnanceur gère les tâches selon l'ordre de priorité et la disponibilité des ressources (sémaphore binaire).
- Le programme contient 3 ressources : Ressource de l'état normale, ressource de l'état alarme, ressource de l'interruption.
- Au début du programme, s'il y'a une présence du gaz on libère la ressource état alarme, si non on libère la ressource état normale. En changement de l'état capteur de gaz une interruption est déclenchée, l'ISR libère la ressource Interrupt ce qui permet à la tâche gaz qui a une priorité élevée de s'exécuter immédiatement. La tâche gaz libère la ressource état normal ou état alarme selon la valeur du capteur du gaz.
- La figure montre ISR donnant un sémaphore binaire à la ressource 2 afin de débloquer cette ressource et mettre la ressource 1 dans un état bloqué, en attendant que le sémaphore recommence à fonctionner.

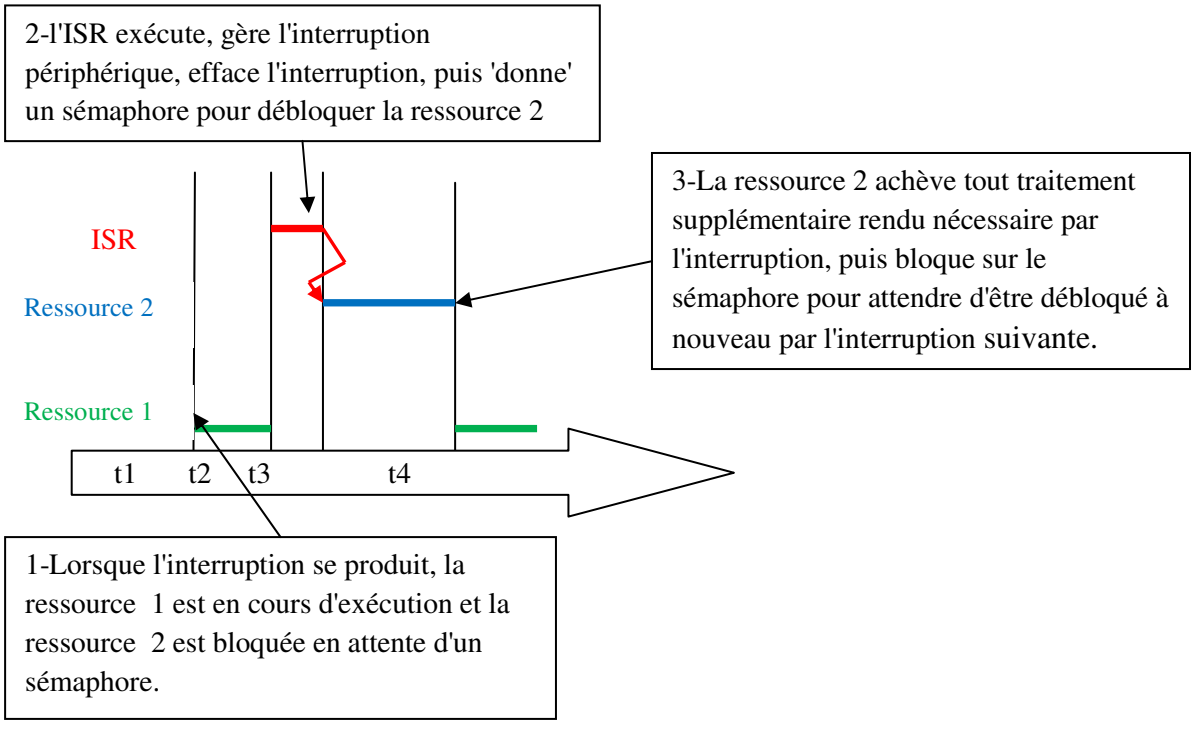


Figure. IV. 8. ISR donnant un sémaphore binaire à la ressource 2.

IV.7. Comparaison et résultats

- La plupart des systèmes d'exploitation semblent permettre à plusieurs programmes ou threads de s'exécuter en même temps. C'est ce qu'on appelle le multitâche. En réalité, chaque cœur de processeur ne peut exécuter qu'un seul programme à un moment donné. Une partie du système d'exploitation appelée ordonnanceur est chargée de décider quel programme exécuter et quand, est fournit l'illusion d'une exécution simultanée en basculant rapidement entre chaque programme.
- Les processus dans FreeRTOS sont appelés les tâches. Chaque processus est un petit programme à part entière. Normalement, une tâche s'exécute indéfiniment dans une boucle infinie et ne doit pas être autorisé à revenir de sa fonction d'implémentation. Les tâches gérées par FreeRTOS sont normalement caractérisés par la taille de la pile et la priorité.
- Le planificateur utilisé dans FreeRTOS fournit le déterminisme en permettant à l'utilisateur d'attribuer la priorité de chaque tâche. Par conséquent, le planificateur gère les tâches en utilisant la priorité pour savoir quelle tâche exécuter. Les tâches ayant la priorité la plus élevée sont exécutées en premier.
- Les systèmes RTOS préemptifs fonctionnent en divisant le temps du processeur en petites tranches et en partageant ces tranches entre des tâches concurrentes. Les systèmes RTOS multitâches coopératifs dépendent de la pause de chaque tâche pour laisser une autre tâche à s'exécuter.

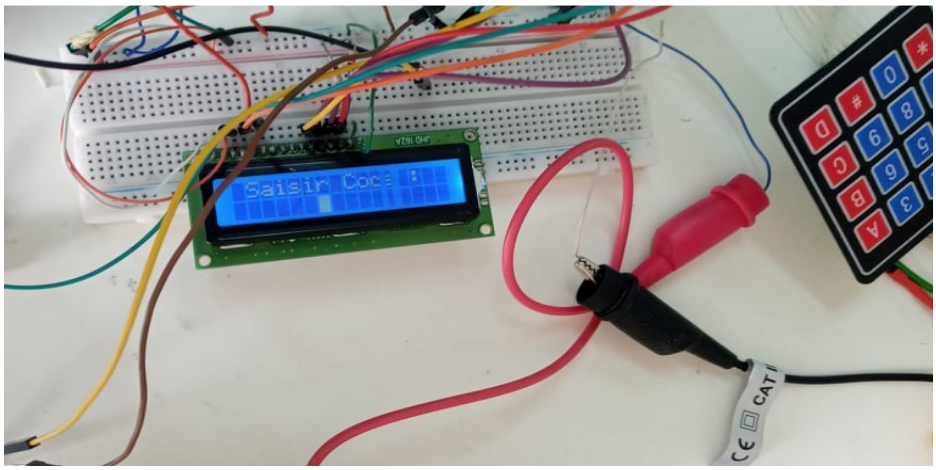


Figure. IV. 9. Photo réel de la vTâche_Porte.

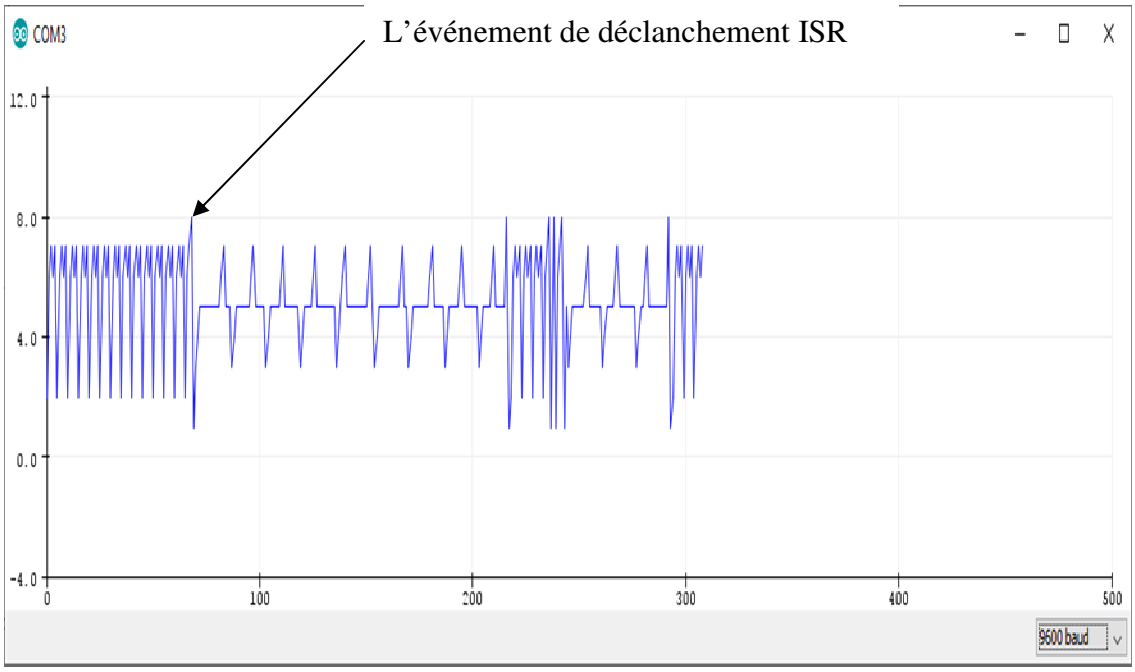


Figure. IV. 10. Capture de traceur série avec FreeRTOS.

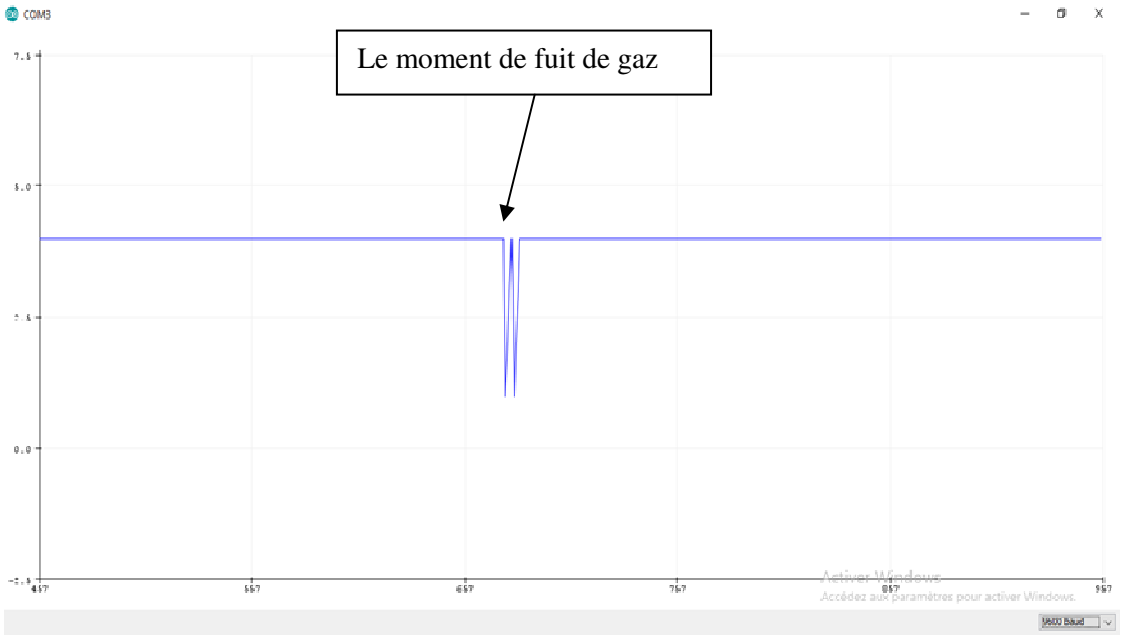


Figure. IV. 11. Capture de traceur série multitâche simple Arduino.

- **Numerisation des tâches :**

L'étude que nous avons mené et l'éxperience qui nous a permet de ce mettre en capacité de pouvoir nomme et nummeroté les tâches selon l'ordre ou nous devons les utilise est qui se présente comme suite :

1 Tâche gaz
2 Led
3 Led Alarme
4 Buzzer
5 Servomotor
6 LDR
7 Afficheur
8 Interruption

Les figures précédents présente deux captures de traceur série dans ARDUINO avec le système FreeRTOS et le système Arduino simple D'après les études que nous avons effectuer , en fait ressortir une remarque , est que le système d'exploitation FreeRTOS utilise la priorité des taches à tout moment c'est a dire , les taches exécutent en parallèle juste au moment de ISR, ou la tache 8 est déjà prête , du moment ou elle a une priorité la plus élevé.

Par contre le système Arduino simple exécute les taches sucssisivment l'une derrière l'autre, ainsi la tache suivante n'est exécuter que si la précédente a était totalement exécuter.

IV.8.Conclusion

Dans ce chapitre, nous avons appliquée des algorithmes sur la plateforme Arduino concernant une maison intelligente. Ainsi dans le projet proposé, il ya eu l'exploitation des algorithmes qui ont été développé en utilisant le noyau open source Arduino Mega et FreeRTOS afin d'accomplir plusieurs tâches. Ces tests ont montrés que les tâches sont exécutées en parallèle, donc le système peut fournir de nombreux services en même temps et les temps de réponses sont instantanés malgré le nombre de tâche.

Conclusion générale

Tout au long de la préparation de notre projet de fin d'études, nous avons essayé de mettre en pratique les connaissances acquises durant nos études universitaires et cela dans le but d'utiliser un système d'exploitation en temps réel RTOS sous la plateforme Arduino pour la gestion de matériel (UAL, Registres, Interruption, Bus…..etc.)Afin d avoir une exécution parallèle multitâche.

Au cours de ce mémoire, nous avons connaitre que l'usage de systèmes d'exploitation est devenue nécessaire dans les systèmes embarqués, du fait de la complexité croissante de ces systèmes, de la présence de fortes contraintes en temps réel de ce la nous avons concentré dans les deux premiers chapitres sur les déférents paramètres de système embarque ensuite nous avons présenté les systèmes d'exploitation en temps réel , dans la troisième chapitre nous avons présenté une classe de RTOS qui est Free RTOS. En fait, le modèle utilisé en programmation des systèmes temps réel est un modèle basé sur la programmation Arduino _ Free RTOS et Arduino simple multitâche.

D'après l'étude que nous avons réalisée, nous remarquons que toutes les applications temps réel RTOS peuvent être définies comme un ensemble des tâches autonomes. Chaque tâche s'exécute dans son propre contexte et ne dépend d'aucune autre tâche. L'ordonnanceur temps réel décide quelle est cette tâche. Il est devra donc indéfiniment démarrer et stopper chaque tâche l'ordonnanceur de RTOS se base uniquement sur la priorité des tâches et concernant l'ordonnanceur multitâche simple Arduino gérer les taches en utilisant la priorité pour savoir quelle tache s'exécutera en suite. D'après l'étude comparative que nous avons réalisé, nous observons que :

- L'ordonnanceur dans le RTOS peut être préventif ou coopératif.
- RTOS permet la création de sémaphores binaires.
- Très faible encombrement mémoire, faible surcharge et exécution très rapide.
- Les tâches peuvent être testées isolément.
- Les codes ne sont exécutés que lorsqu'un événement doit être effectué.
- Les tâches sont soit non bloquantes, soit bloquées après une période de temps fixe.
- Aucun temps de traitement n'est perdu en interrogeant des événements qui ne se sont pas produits.
- Lancer plusieurs exécutions du même code simultanément.
- Exploiter la structure multiprocesseur des ordinateurs moderne.

Conclusion générale

Malgré l'importance et les avantages de RTOS, il a aussi des inconvénients, les système d'exploitation et sa mise en œuvre compliquée, de plus il consomme le stockage mémoire RAM et ROM en raison de la faible capacité de microcontrôleur ATMega.

Enfin on peut conclure qu'il existe un domaine très riche impliquant des applications RTOS pour les taches embarquées. Donc il est possible de mette en place un environnement agréable et confortable pour procéder à la conception et à l'étude des systèmes temps réels vu le nombre, la simplicité et la puissance d'outils qui dont à notre disposition actuellement.

Bibliographies

[1]- Kara Abdelaziz. Mémoire de master « Le développement d'un système embarqué implémentant le Peer-to-Peer pour la TVoIP » UNIVERSITE FERHAT ABBAS – SETIF V ECOLE DOCTORALE D'INFORMATIQUE (STIC).

[2]-Système temps réel *Responsable de Cours :* **Hugo Descoubes**, hugo.descoubes@ensicaen.fr. Spécialité Électronique et Physique Appliquée-2015/2016.

[3]- M. Barr. Embedded Systems Glossary. NetrinoTechnical Library, www.netrino.com.

[4]-Embedded Systems: Real-Time Interfacing to Arm® Cortex™-M Microcontrollers (Anglais) Broche – 10 November 2011 de Jonathan W. Valvano (Auteur)

[5]-BENDIB Ahmed. Mémoire de master « Evaluation de la consommation en énergie des programmes implantés dans les systèmes embarqués à base de microprocesseurs DSPs » UNIVERSITE MOHAMMED BOUDIAF-M'SILA V Commande des systèmes électro-énergétiques-2014. Consulté (18-07-2020)

[6]-https://er.yuvayana.org/classification-of-embedded-system-with-details/ .Consulté (19-07-2020)

[7]-Real–Time Systems Design and Analysis: Tools for the Practitioner (Anglais) Relié – 23 décembre 2011 de Phillip A. Laplante (Auteur), Seppo J. Ovaska (Auteur)

[8]- A- Doncescu. Introduction aux Systèmes Embarqués et Microcontrôleurs. [En Ligne]. Disponible sur : https://homepages.laas.fr/adoncesc/SystemEmbed/Cours1-introd.pdf . Consulté (19-07-2020)

[9]- L- Pautet. Introduction aux Systèmes embarqués temps-réel. [En Ligne]. Disponible sur : http://perso.telecom-paristech.fr/~pautet/seti/supports/rt-intro.pdf . Consulté (22-07-2020)

[10]--H. Hansson, M. Nolin, T. Nolte. *Embedded systems handbook*. CRC Press, Editeur: Richard Zurawski, 2005.

[11]- J. Stankovic. Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems, *IEEE Computer,* Vol. 21, No. 10, Octobre 1988

[12]-https://www.universalis.fr/encyclopedie/systemes-d-exploitation-informatique/ Consulté (22-06-2020)

[13]-Hands-On RTOS with Microcontrollers: Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools (English Edition) Format **Kindlede Brian Amos** (Auteur)

[14]-Why RTOS and What Is RTOS? Available online: https://www.freertos.org/about-RTOS.html. Consulté (22-06-2020)

[15]-Real-Time Concepts for Embedded Systems. Editor : **Qing Li** with **Caroline Yeo**.

[16]-Distributed real-time operating system (DRTOS) modeling in SpecC. Disponible sur ore.ac.uk/download/pdf/38896907.pdf

[17]- General RTOS Concepts. Disponible sur : www.renesas.com. Consulté (02-08-2020)

[18]-https://www.guru99.com/real-time-operating system. Consulté (02-08-2020)

[19]-https://www.engineersgarage.com/article_page/rtos-real-time-operating-system/. Consulté (02-08-2020)

[20]- https://freertos.org/. Consulté (15-08-2020)

[21]- Microprocesseurs et Microcontrôleurs. Jlassi Khaled . Disponible sur : http://pf-mh.uvt.rnu.tn/320/1/Microprocesseurs_et_Microcontr%C3%B4leurs.pdf. Consulté (15-08-2020)

[22] -Maîtriser le noyau en temps réel de FreeRTOS. Consulté (15-08-2020)

[23]-Microcontrollerslab.com/. Consulté (15-08-2020)

[24]- Manuel de référence FreeRTOS V10.0.0. Consulté (15-08-2020)

[25]- Patrick Renard Acquisition de **données** du capteur, Georges, Asch, E, chambard, Gunther 528p, (2003), Dunod.

[26]- Arduino Mega 2560 roboromania. Disponible sur: https://roboromania.ro/datasheet/Arduino-Mega-2560-roboromania.pdf. Consulté (03-01-2021)

[27]-Piezoelectronic Buzzer PS serie. Disponible sur: https://product.tdk.com/info/en/catalog/datasheets/piezoelectronic_buzzer_ps_en.pdf. Consulté (03-01-2021).

[28]-Datasheet MQ-2. Disponible sur : https://fr.hobbytronics.co.uk/mq2-gas-smoke-sensor.pdf. Consulté (03-01-2021).

[29]-Light dependent resistor Datasheet. Disponible sur : https://components101.com/sites/default/files/component_datasheet/LDR%20Datasheet.pdf. Consulté (03-01-2021).

[30]-Servomoteur Datasheets. Disponible sur : https://www.datasheetarchive.com/servomoteur-datasheet.html. Consulté (03-01-2021).

[31]- 16X2 LCD Datasheet. Disponible sur : https://components101.com/16x2-lcd-pinout-datasheet. Consulté (03-01-2021).

[32]-LED Datasheets. Disponible sur : http://www.farnell.com/datasheets/1498852.pdf. Consulté (03-01-2021).

[33]- http://www.composelec.com/resistance_(composant).php. Consulté (03-01-2021).

[34]http://www.zpag.net/Electroniques/relais.htm#:~:text=Un%20relais%20est%20un%20appareil,affaire%20%C3%A0%20un%20relais%20statique). Consulté (03-01-2021).

[35]-https://www.electronicwings.com/arduino/4x4-keypad-interfacing-with-arduino-uno. Consulté (03-01-2021).

[36]- http://www.elektronique.fr/logiciels/proteus.php. (05-01-2021).

Résume

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.

Parmi les RTOS le plus utilisé est le FreeRTOS. FreeRTOS est un système d'exploitation embarqué multitâches temps réel préemptif supporte actuellement 35 architectures. Il est aujourd'hui parmi les plus utilisés dans le marché des systèmes d'exploitation temps réel.

L'objectif de ce travail est l'utilisation d'un système d'exploitation en temps réel RTOS sous la plateforme Arduino pour la gestion des ressources matériel afin d'avoir une exécution parallèle de multitâches.

Mot clé : Système embarqué, système temps réel, RTOS, FreeRTOS, Arduino.

Abstract

Embedded systems are becoming more and more complex. It is for this reason that designers resort to the use of a multitasking real-time kernel (RTOS) to handle this complexity.

Among the most used RTOS is the FreeRTOS. FreeRTOS is a preemptive real-time multitasking embedded operating system currently supporting 35 architectures. It is among the most widely used in the real-time operating system market today.

The objective of this work is the use of an RTOS real-time operating system under the Arduino platform for the management of hardware resources in order to have parallel execution of multitasking.

Keyword: Embedded system, real time system, RTOS, FreeRTOS, Arduino.