Systèmes temps Réel

chapitre 0 (Introduction)

1) Introduction

- En informatique, il y a deux parties :
  1- La partie Hard (microprocesseur) MP ou MC.
  2- La partie Soft (logiciel) - programmation
- pour la partie Soft, on va utiliser un Système d'Exploitation (OS) - qui peut être soit :
  - Windows
  - Linux
  - Android
  - .....
- En anglais, le Système d'Exploitation OS est appelé : Operating System OS.
- * Parmis les OS, il y a un type de Système d'Exploitation appelé : Système Temps réel (STR), en Anglais, on l'appel : Réal Time operating System. (RTOS).

2) Définition :

- un RTOS est un Système Informatique utilisé pour commander / communiquer / traiter un processus réel (moteur, clavier, imprimante ...) en temps réel.
- Exemple : si on fais de la simulation avec le logiciel Matlab, dans ce cas, Matlab n'est pas un RTOS mais seulement un OS.

* Par contre, le Système qui gère un feu Vert pour la conduite des voitures est un RTOS car ce Système gère des processus réel (des voitures).

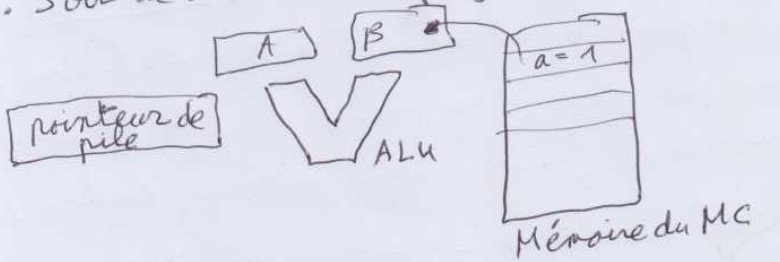
* un Autre exemple de RTOS est le logiciel Matlab lorsque ce logiciel est utilisé pour commander en temps réel un moteur électrique (utilisation de PC + carte d'acquisition).

• Dans ce cours de Systèmes Temps réel on va toujours parler de la partie Soft, on va rarement parler de la partie Hard (MC).

3) Revision - la pile du MS. (Stack en anglais)

. Pour commencer de parler sur les RTOS, leurs utilisations, leurs fonctionnements, on va d'abord rappeler le rôle du pointeur de pile pour le fonctionnement d'un Microcontroleur. On aura besoin plus tard de cette notion de pointeur ...

. Soit le schema simplifié suivant d'un Microcontroleur



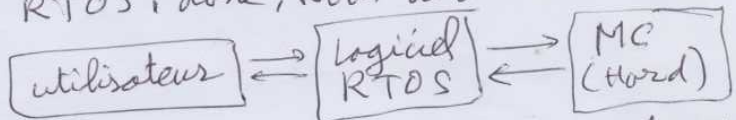
* pour fonctionner, le MC a besoin d'une mémoire, d'une ALU : unité arithmetique et logique, de deux registres de donnée A et B, et du pointeur de pile

- le pointeur de pile est un registre qui contient à chaque moment, l'instruction en cours, que va réaliser l'ALU (l'instruction peut etre soit une saisi d'une valeur numerique, ou bien un opération à effectuer).
- Apres avoir rappelé la notion de pointeur de pile, on va passer à définir les Systemes temps réel &

---

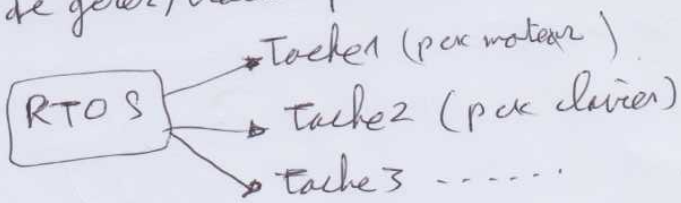
4) RTOS - les taches:

* Maintenant on va supposer qu'on est entrain d'utiliser un RTOS, donc, Nous avons le schema suivant:



→ l'utilisateur va communiquer avec le MC à travers le Systeme informatique RTOS.

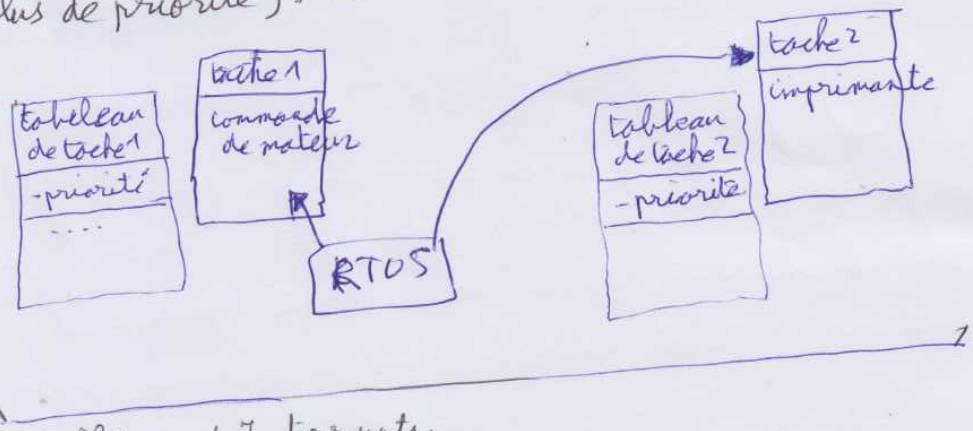
° generalement, le RTOS est utilisé lorsque le MC a besoin de gérer/traiter plusieurs taches (processus réels)



° à chaque moment, le RTOS doit communiquer avec les taches et il dois selectionner un seule parmis ces taches pour la lancer,

* Donc, la question qui se pose, Quelle est la tache que dois lancer (commander) le RTOS à chaque

- Pour repondre a cette question, le Système informatique RTOS qu'on est entrain d'utiliser, doit communiquer avec ces taches pour connaitre quel est l'état de chaque tache (marche ou arrêt?), et ceci à chaque horloge du MC.
- Lorsque le RTOS recoit les informations sur l'état de chaque tache, ensuite il va verifier à chaque moment le tableau de chaque tache pour savoir quelle est la tache la plus prioritaire (celle qui a plus de priorité). voir schema suivant:

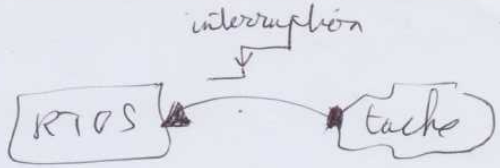


5) Pollings et Interrupts

- Pour qu'un OS ou RTOS communique avec des taches (processus) il y a deux possibilités, soit par pollings ou par des interruptions (Interrupts en anglais).

a) Méthode Pollings

Dans cette methode, à chaque Horloge, le RTOS envoie une impulsion (un signal) à chaque tache, pour verifier l'état de cette tache (marche = 1, arret = 0). (RTOS) → (tache



b) Methode Interrupts :

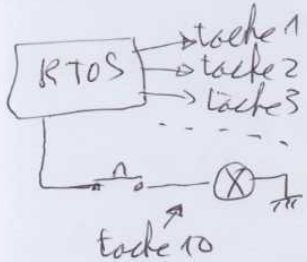
- Dans cette methode, c'est la tache qui envoie au RTOS une interruption (un signal), pour interrompre le RTOS (donc aussi le MC), et lui demander de lancer cette nouvelle tache et arreter la tache que le RTOS etait entrain de traiter.
- Le RTOS, pour savoir s'il lance la nouvelle tache ou bien il continue de traiter l'ancienne tache, il va verifier la priorité de chaque tache.

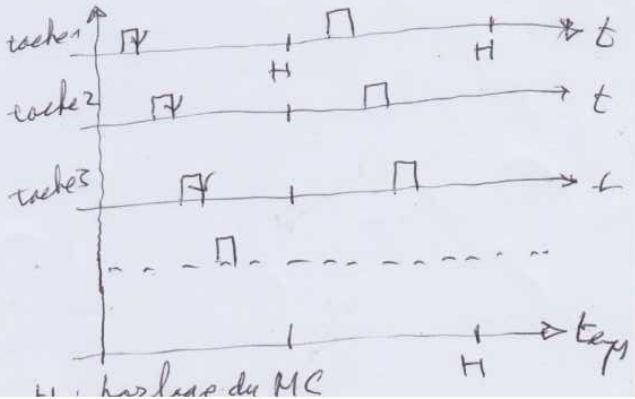
Question :

- Pour un Systeme RTOS, qu'elle est la meilleur methode de communication avec les taches? Pollings ou bien Interrupts?

Réponse:

pour repondre à cette question (par polling) on va supposer que nous avons un RTOS qui gère 10 taches, et parmis ces taches on a la commande d'une LED par bouton poussoir

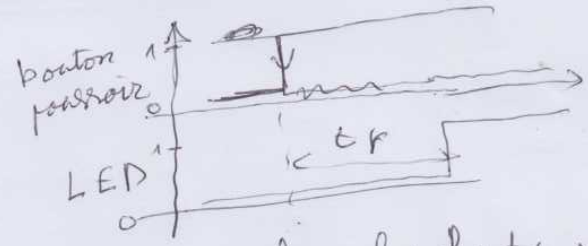




RTOS avec méthode Polling

H : horloge du MC

- on va supposer qu'on utilise la methode polling : donc, à chaque horloge H, le RTOS va envoyer 9 signaux de verification d'état pour les 9 taches, en plus de la 10ème tache qui est le bouton poussoir - LED
- ces signaux doivent etre decalé car le MC fait 1 seule ~~tache à la~~ operation à la fois,
- envoyer 9 signaux à chaque horloge du MC, ceci va occuper le fonctionnement du MC, donc, le MC pour allumer la LED (une fois le bouton poussoir fermé manuellement) va repondre avec un retard tr



- plus il y a de taches, plus de tr sera plus grand, ceci est un inconvenient majeur pour la methode polling
- Donc pour les RTOS on n'utilise pas la methode polling mais la methode Interrupts, ou, à chaque horloge, un seul signal sera envoyé par la tâche qui veut se lancer.
- Dans la methode ~~RTOS~~, Interrupts, le RTOS n'envoie pas de signal de verification aux taches mais c'est la tache qui envoie un signal d'interruption

7

Soit le pseudo code suivant pou les deux cas, Polling et Interrupts, pour le cas Polling o, voie que le programme (RTOS) envoie à chaque fois un signal de vérification Check alors que pour la méthode Interrupts nous avons les taches qui envoient des interruptions

Polling

```
int main(void) {
        while(1){

        //Task 1

        //Check PF0
            if(PF0 == 1){
            //Task 2
            }

        //Check Timeout Flag
            if((Timer0_RIS & 0x01) == 1){
            //Task 3
            Timer0_ICR |= (1<<0);
            }

        }
}
```

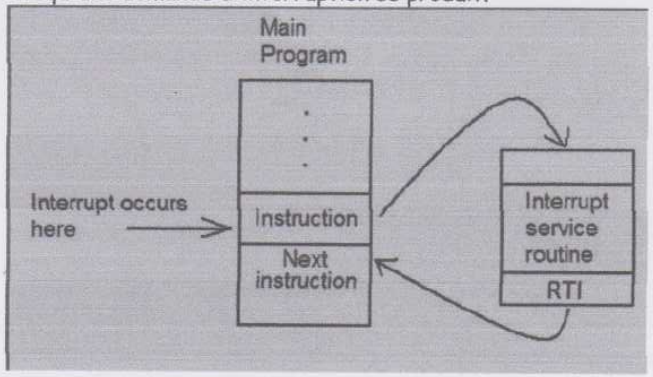
Interrupt

```
int main(void) {
        while(1){
        //Task 1
        }
}

void GPIOF_Inerrupt(void){
        //Task 2
}

void timer0_inerrupt(void){
        //Task 3
        Timer0_ICR |= (1<<0);
}
```

La figure suivante montre le microprocesseur exécutant son programme principal lorsqu'une demande d'interruption se produit.



En réponse à la demande d'interruption, le microprocesseur

1. Termine l'exécution de l'instruction en cours.
2. Accède à la routine du service d'interruption, où il effectue toutes les actions sont requis par l'interruption particulière.
3. Retourne au programme principal et reprend le traitement au prochaine instruction.

On peut considérer l'interruption comme une autre tache (tache2), l'interruption peut être aussi un évènement inattendu.