TP3 –Conception des systèmes temps réels
Notion de sémaphore

I. Objectif :

- Concevoir une application temps réel en se basant sur la notion de sémaphore.

II. Notion des sémaphores

Un sémaphore est un mécanisme empêchant deux processus ou plus d'accéder simultanément à une ressource partagée. Par exemple sur les voies ferrées, un sémaphore empêche deux trains d'entrer en collision sur un tronçon de voie commun. Dans ce sens un processus doit tester un sémaphore avant d'accéder à une ressource partagée afin d'éviter toute collision possible

Un sémaphore binaire se distingue par deux états :

- Etat : 0 verrouillé (ou occupé)
- Etat : 1 déverrouillé (ou libre)

Un sémaphore général peut avoir un très grand nombre d'états car il s'agit d'un compteur dont la valeur initiale peut être assimilée au nombre de ressources disponibles. Par exemple, si le sémaphore compte le nombre d'emplacements libres dans un tampon et qu'il y en ait initialement 5 on doit créer un sémaphore général dont la valeur initiale sera 5. Ce compteur :

- Décroît d'une unité quand il est acquis (" verrouillé ").
- Croît d'une unité quand il est libéré (" déverrouillé ").

Quand il vaut zéro, un processus tentant de l'acquérir doit attendre qu'un autre processus ait augmenté sa valeur car il ne peut jamais devenir négatif. L'accès à un sémaphore se fait généralement par deux opérations :

P : pour l'acquisition

V : pour la libération

Un moyen mnémotechnique pour mémoriser le P et le V est le suivant : – P(uis-je) accéder à une ressource. – V(as-y) la ressource est disponible.

Dans le noyeau OSA, tous les sémaphores binaires ont une taille d'un bit. Le programmeur peut définir le nombre de sémaphores binaires en définissant la constante OS_BSEMS dans

OSAcfg.h et le système allouera le nombre approprié d'octets. Par conséquent, la mémoire des sémaphores binaires est réservée au moment de la compilation et leur nombre ne peut pas être modifié au moment de l'exécution.

Lors de l'utilisation de services contrôlant des sémaphores binaires, il est nécessaire de fournir l'ID du sémaphore (entre 0 et OS_BSEMS-1).Par exemple :

```
enum OSA_BSEMS_ENUM
{
        BSEM_WRITE
};
```

Une tâche qui attend un sémaphore binaire sera placée dans un état d'attente jusqu'à ce que le sémaphore binaire soit défini.

OS_Bsem_Wait（BSEM_WRITE） : marque l'attente de la libération d'un sémaphore

OS_Bsem_Set（BSEM_WRITE）: marque la libération de la ressource

III. Travail demandé :

Nous allons réaliser dans cette partie une application à base du noyau OSA munie de deux tâches sur un microcontrôleur 18F4550.

1) Créer un nouveau projet OSA et ajouter le noyau temps réel OSA.
2) Ecrire un programme qui permet de communiquer l'état des LEDs sur un PC à travers une liaison série USART.

Tâche 1 : renvoie l'état allumé de deux LEDs connectées respectivement sur le bit 1 et 2 du PORTD.

Tâche 2 : renvoie l'état éteint de deux LEDs connectées respectivement sur le bit 1 et 2 du PORTD.

```
#include <osa.h>
#include <OSAcfg.h>
#pragma funcall main Tache2
#pragma funcall main Tache1
void InitTimer0(){
  T0CON        = 0x88;
  TMR0H        = 0xD1;
  TMR0L        = 0x21;
  GIE_bit      = 1;
```

```
 TMR0IE_bit    = 1;
}
void Interrupt(){
 if (TMR0IF_bit){
   TMR0IF_bit    = 0;
   TMR0H         = 0xD1;
   TMR0L         = 0x21;
   OS_Timer();  }     // incrémentation du timer
}

   void Tache1(void)
{
     while(1) {
       UART1_Write_Text("task1: leds allumées ");
      UART1_Write(13);
      UART1_Write(10);
                    // changer l'état du bit1 du portd
                   // changer l'état du bit2 du portd
                 }
}
      void Tache2(void)
{
     while(1)     {
      UART1_Write_Text("task2: leds éteintes ");
      UART1_Write(13);
      UART1_Write(10);
                      // changer l'état du bit1 du portd
                     // changer l'état du bit2 du portd
     }
}

void main() {
```

```
// configurer le portd en sortie
//initialiser le portd avec zéro
 OSCCON =  0X70;    // internal oscillator to 8MHz
ADCON1 = 0x0F;     // Configurer AN pins
CMCON = 7;         //  désactiver le comparateur
UART1_Init(19200);           // Init USART module

 OS_Init();

 OS_Task_Create(0,Tache2);
OS_Task_Create(0,Tache1);
InitTimer0();
 OS_Run();
}
```

3) Réaliser un schéma ISIS et tester votre application.