TP2 –Conception des systèmes temps réels
Notion des priorités

I. Objectif :

- Concevoir une application temps réel en se basant sur la notion des tâches et leurs priorités.

II. Notion des tâches et priorités

Toutes les tâches ont une priorité de 0 (le plus élevé) à 7 (le plus bas). Il existe trois modes de priorité :

- Priorités désactivées (disabled) : C'est un mode rapide et compact où toutes les priorités assignées sont ignorées. En revanche, il ne permet pas d'attribuer une priorité plus élevée pour des tâches plus importantes.
- Priorités normales (normal priorities) : Chaque tâche a sa propre priorité. S'il y a plusieurs tâches prêtes, alors la tâche avec la plus haute priorité sera exécutée. Si plusieurs tâches prêtes ont la même priorité, alors elles seront fonctionnées en mode « round-robin ». Ce mode présente deux inconvénients:

-Quand il y a une tâche toujours prête avec une haute priorité, aucune tâche de faible priorité ne sera exécutée.

- Lorsque deux ou plusieurs tâches ont la même priorité et elles attendent pour le même événement, une seule d'entre elles obtiendra le contrôle.

- Priorités étendues (extended priorities) : toutes les tâches prendront le contrôle en fonction de leurs priorités.

Par exemple, s'il y a deux tâches toujours prêtes avec les priorités 3 et 4, puis l'une d'entre elles obtiennent 55-60% de contrôle (avec priorité 4) et d'autres - 40-45%.

L'avantage de ce mode garantit que toutes les tâches auront le contrôle. Mais dans ce système en mode nécessite 2 octets d'addition de RAM pour chaque tâche.

Le mode de priorité est défini une fois sur la scène de la compilation par la configuration du paramètre OS_PRIORITY_LEVEL dans osacfg.h:

#define OS_PRIORITY_LEVEL OS_PRIORITY_DISABLE // Pour désactiver les priorités

#define OS_PRIORITY_LEVEL OS_PRIORITY_NORMAL // Pour priorités normales

#define OS_PRIORITY_LEVEL OS_PRIORITY_EXTENDED // Pour priorités étendues

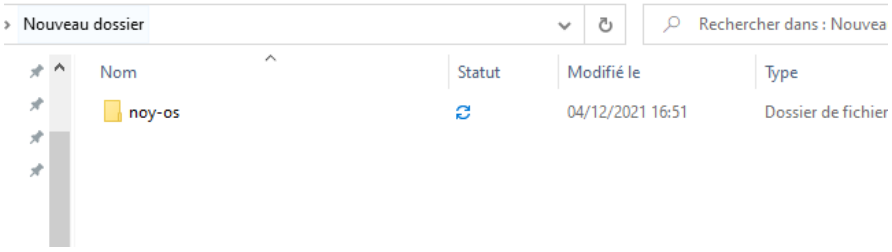
Par défaut OS_PRIORITY_NORMAL est réglé.

III. Travail demandé :

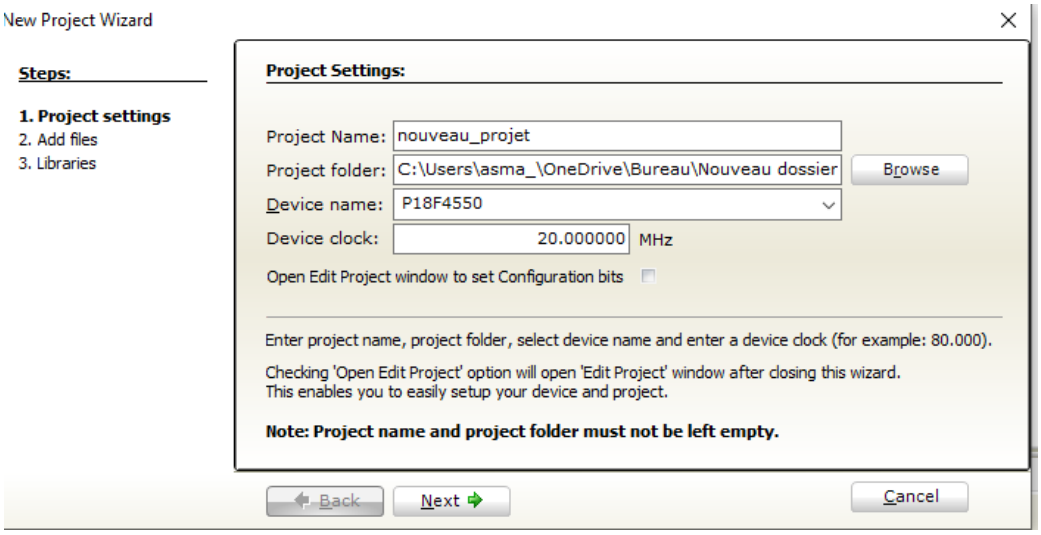
Nous allons réaliser dans cette partie une application à base du noyau OSA munie de deux tâches sur un microcontrôleur 18F4550.

1) Suivez les étapes suivantes pour réaliser une application temps réel.

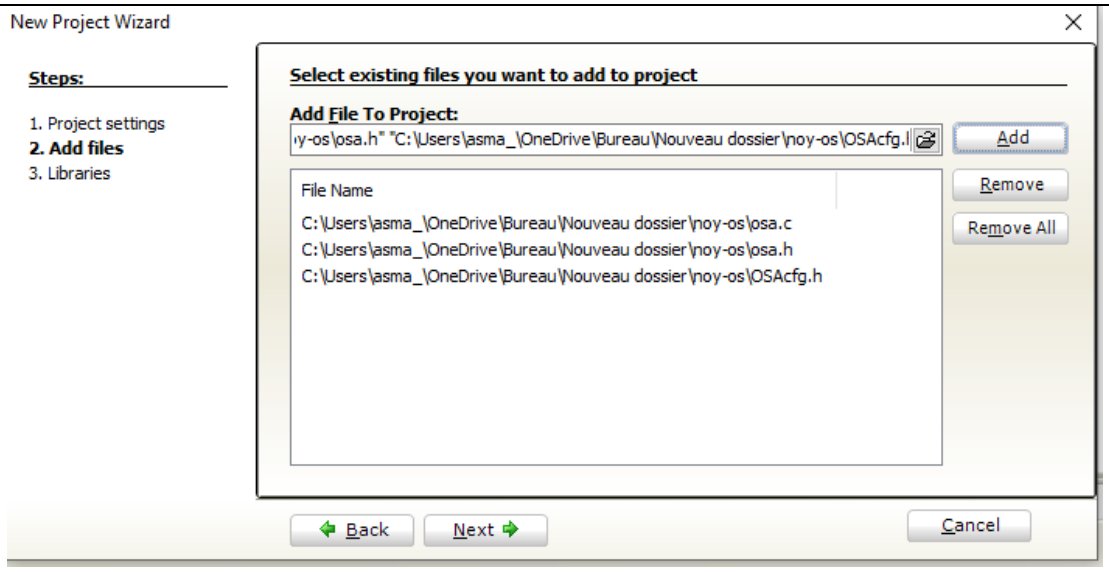
- Créer un nouveau dossier sur le bureau et le renommer sous le nom de votre projet.
- Importer le dossier noyau_OSA dans ce nouveau dossier



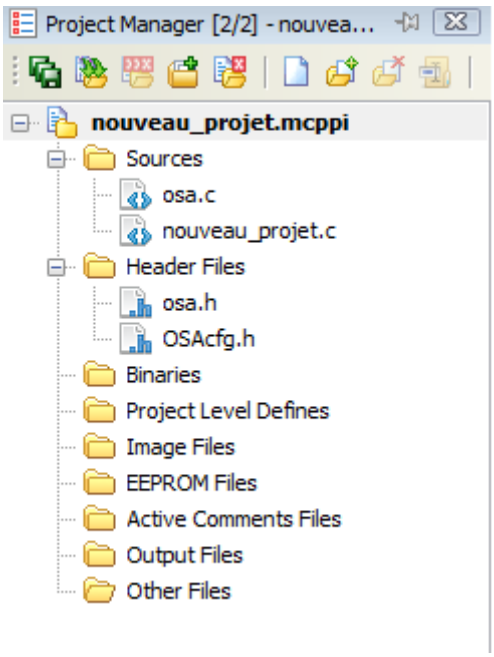
- Créer un projet MikroC et le sauvegarder sous le même chemin du nouveau dossier créé.



- Ajouter les fichiers osa.c, osa.h, oscfg.h



- Vérifier les fichiers ajoutés dans l'arborescence du projet



2) Ecrire un programme qui permet de créer deux tâches, une tâche qui fait allumer une led connectée sur le bit 1 du PORTD et une autre tâche qui fait allumer une autre led connectée sur le bit 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");
      UART1_Write(13);
      UART1_Write(10);
                    // changer l'état du bit1 du portd
       OS_Yield();
                  }
}
      void Tache2(void)
{
     while(1)     {
      UART1_Write_Text("task2");
      UART1_Write(13);
      UART1_Write(10);
                      // changer l'état du bit2 du portd
```

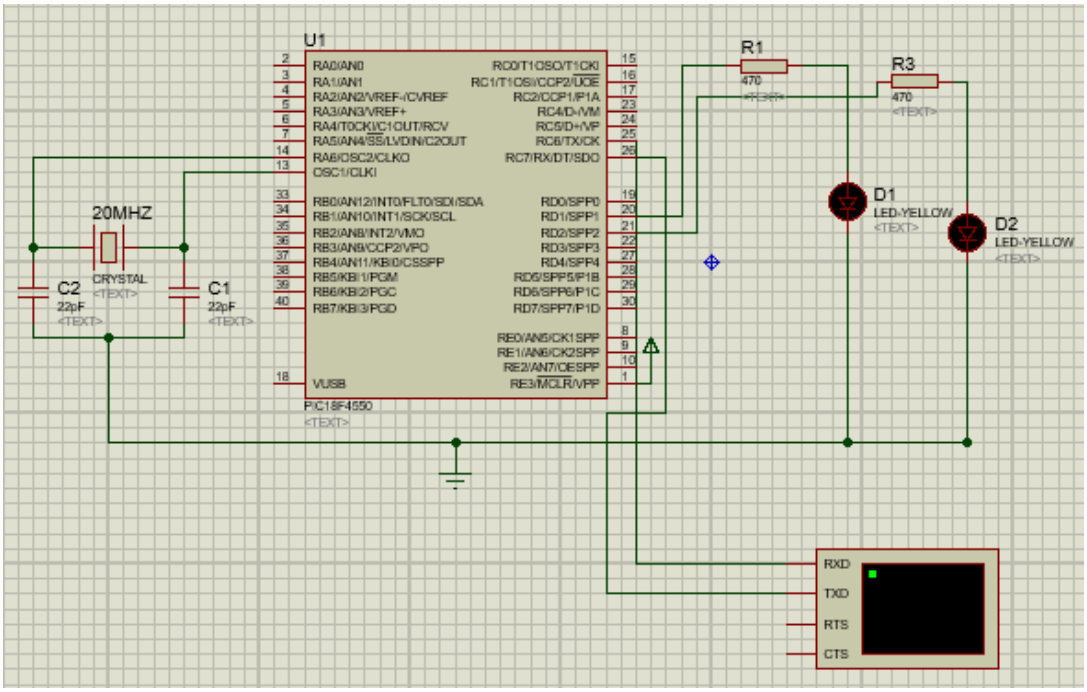
```
        OS_Yield();
       }
}

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.



4) Changer le type de priorité dans osacfg.h puis remplir le tableau ci-dessous pour chacun des cas suivants :

	Tâche en cours d'exécution	
	Priorité normale	**Priorité étendue**
Tâche 1 plus prioritaire que la tâche 2		
Tâche 2 plus prioritaire que la tâche 1		
Tâche 1 et la tâche 2 ont le même niveau de priorité mais la tâche 1 est créée avant la tâche 2		
Tâche 1 et la tâche 2 ont le même niveau de priorité mais la tâche2 est créée avant la tâche1		
Tâche 1 plus prioritaire que la tâche 2 et on introduit un delay de 500 ms (OS_Delay(500))		
Tâche 1 plus prioritaire avec OS_Delay(500) la tâche 2 avec un retard (OS_Delay(100))		