TP FreeRTOS

Partie 0: Reprise en main

1. Le fichier main.c se situe dans les fichiers Core -> Src.

2. Les balises servent de repères, si l'on ajoute du code en dehors de ces balises il sera supprimer si l'on utilise le .ioc pour générer du code.

3. Le code des fonctions ressemble à ceci : `HAL_GPIO_WritePin(GPIOx, GPIO_Pin, PinState)` et `HAL_Delay(Delay)`.

Il faut donc donner en paramètre un int à HAL_Delay qui représente la durées en millisecondes. Et il faut donner le port GPIO puis le Pin sur lequel on souhaite agir à la fonction HAL_GPIO_WritePin ainsi que l'état que l'on souhaite donné (1 ou 0).

4. Les ports d'entrées/sorties sont définis dans le fichier gpio.c qui se trouve au même endroit que le fichier main.c.

Partie 1: FreeRTOS, tâches et sémaphores

1. Le paramètre TOTAL_HEAP_SIZE représente la taille du tas. Il contient entre autre les piles des taches il faut donc le choisir suffisamment grand pour qu'il ne soit pas remplis.
2. Pour faire clignoter la led toutes les 100ms on réalise le code suivant :

```
void vTaskLED( void * pvParameters )
  {
      for( ;; )
      {
                HAL_GPIO_TogglePin(GPIOI, LED_Pin);
              printf("changement led");
                vTaskDelay(100);
      }
  }
```

   Le rôle de la macro portTICK_PERIOD_MS qui se trouve en paramètre de vTaskDelay est définir le nombre de Tick durant lequel la tache va s'endormir (ici 1 tick = 1ms).

3. On réalise le code suivant :

```
void vTaskTake( void * pvParameters )
  {
      for( ;; )
      {
          printf("Task va prendre le semaphore\r\n");
          xSemaphoreTake(xSemaphore1 ,portMAX_DELAY );
            printf("Semaphore pris: \r\n");


      }
  }

void vTaskGive( void * pvParameters )
  {

      for( ;; )
      {       printf("Task va donner le semaphore\r\n");

              xSemaphoreGive( xSemaphore1 );
              printf("Semaphore donné\r\n");
              vTaskDelay(100);

              }

      }
```

TaskGive a une priorité de 3 et TaskTake une priorité de 2. On obtient alors ce résultat :

```
Task va prendre le semaphore
Task va donner le semaphore
Semaphore donné
Semaphore pris:
Task va prendre le semaphore
```

C'est le résultat après la première itération. TaskTake veut prendre le sémaphore puis elle bloque en attendant que TaskGive donne le sémaphore qui s'endort ensuite pour 100ms on revient alors à TaskTake qui affiche qu'elle a pris le sémaphore puis rebloque en attendant le prochain. Pour le premier cycle, c'est TaskGive qui est prioritaire on aurait donc : Task va donner -> Sem donné -> Task va prendre -> Sem pris ; puis on arriverait au résultat ci-contre.

4. Pour ajouter un mécanisme de gestion d'erreur il faut réaliser le code suivant :

```
void vTaskTake( void * pvParameters )
{

    for( ;; )
    {
        printf("Task va prendre le semaphore\r\n");
        if(xSemaphoreTake(xSemaphore1 ,1000) == pdTRUE)
            {
            printf("Semaphore pris: \r\n");
                }
        else{
            printf("reset\r\n");
           NVIC_SystemReset();
           i=0;
        }

    }
}

vTaskGive( void * pvParameters )
{

    for( ;; )
    {       printf("Task va donner le semaphore\r\n");

            xSemaphoreGive( xSemaphore1 );
            printf("Semaphore donné\r\n");
            printf("i= %d",i);
        vTaskDelay(i);
        i=i+100;


    }
}
```

On met à 1000 la valeur du paramètre *xTicksToWait* de la fonction xSemaphoreTake et on test si elle nous renvoie bien pdTRUE. Ce qui veut dire que si la tâche attend plus de 1000ms le sémaphore on n'obtiendra pas pdTRUE et on réalisera un SystemReset. Dans TaskGive on augmente le délai de 100ms à chaque cycle. Et voici ce que l'on obtient :

```
Task va prendre le semaphore
Task va donner le semaphore
Semaphore donné
i= 1100Semaphore pris:
Task va prendre le semaphore
reset
Task va donner le semaphore
Semaphore donné
i= 100Task va prendre le semaphore
Semaphore pris:
Task va prendre le semaphore
Task va donner le semaphore
Semaphore donné
i= 200Semaphore pris:
Task va prendre le semaphore
Task va donner le semaphore
```

Lorsque le délai i=1100ms et que l'on atteint le xSemaphoreTake on effectue bien un reset puis on retrouve le comportement expliqué précedement. *(Je ne sais pas pourquoi mais si je rajoute \r\n dans le printf du i le code ne fonctionne plus).*

6. Si on échange les priorités voilà ce qui s'affiche :

```
Task va prendre le semaphore
Task va donner le semaphore
Semaphore pris:
Task va prendre le semaphore
Semaphore donné
i= 100Task va donner le semaphore
Semaphore pris:
Task va prendre le semaphore
Semaphore donné
i= 200Task va donner le semaphore
Semaphore pris:
Task va prendre le semaphore
Semaphore donné
```

On a deux fois Task va prendre le semaphore avant que Semaphore donné s'affiche. C'est normal puisque TaskTake étant maintenant prioritaire elle reprend la main dès que la ligne xSemaphoreGive est réalisée et effectue donc une nouvelle boucle avant de rendre la main à TaskGive qui reprend à la ligne Semaphore donné.

7. Pour utiliser des notifications à la place des sémaphores il faut simplement remplacer les ligne xSemaphoreTake/Give par leurs équivalents avec des notifications :

```
void vTaskTake( void * pvParameters )
    {

        for( ;; )
        {
            printf("Task va prendre le semaphore\r\n");
            //if(xSemaphoreTake(xSemaphore1 ,1000) == pdTRUE)
                if(ulTaskNotifyTake(pdTRUE, 1000) == pdTRUE)
                    {
                printf("Semaphore pris: \r\n");
                    }
            else{
                printf("reset\r\n");
               NVIC_SystemReset();
               i=0;
            }

        }
    }

vTaskGive( void * pvParameters )
    {

        for( ;; )
        {       printf("Task va donner le semaphore\r\n");
                //xSemaphoreGive( xSemaphore1 );
                xTaskNotifyGive(xHandle3);
                printf("Semaphore donné\r\n");
                printf("i= %d",i);
            vTaskDelay(i);
            i=i+100;


        }
    }
```

On a alors exactement le même comportement que précédemment. Il faut cependant maintenant déclare le xHandle3 (qui est le Handle de TaskTake) avant le code des tâches et non plus au début de la boucle main.

8. Pour utiliser des queues il faut réaliser le code suivant :

```
vTaskGive( void * pvParameters )
{
  char TxTimerBuff[100];
  xQueue1 = xQueueCreate(10, sizeof(TxTimerBuff));
    for( ;; )
    {       printf("Task va donner le semaphore\r\n");
            //xSemaphoreGive( xSemaphore1 );
    sprintf(TxTimerBuff, "%d",i);
        xQueueSend(xQueue1, (void*) TxTimerBuff,(TickType_t) 0);

            xTaskNotifyGive(xHandle3);
            printf("Semaphore donné\r\n");
           // printf("i= %d",i);
        vTaskDelay(i);
        i=i+100;
    }
}
```

```
void vTaskTake( void * pvParameters )
    {

        for( ;; )
        {
            char RxTimerBuff[100];
            printf("Task va prendre le semaphore\r\n");
            //if(xSemaphoreTake(xSemaphore1 ,1000) == pdTRUE)
                if(ulTaskNotifyTake(pdTRUE, 1000) == pdTRUE)
                    {
                printf("Semaphore pris: \r\n");
                if(xQueueReceive(xQueue1,(void*)RxTimerBuff, (TickType_t)5))
                      {
                      printf("timer: %s\r\n", RxTimerBuff);
                          }
                    }
            else{
                printf("reset\r\n");
               NVIC_SystemReset();
               i=0;
            }
        }
    }
```

On obtient alors toujours le même résultat mais cette fois ci l'affichage de la valeur du délai ce fait dans la tâche TaskTake :

```
vTaskGive( void * pvParameters )
  {
    char TxTimerBuff[100];
    xQueue1 = xQueueCreate(10, sizeof(TxTimerBuff));
      for( ;; )
      {       printf("Task va donner le semaphore\r\n");
              //xSemaphoreGive( xSemaphore1 );
      sprintf(TxTimerBuff, "%d",i);
          xQueueSend(xQueue1, (void*) TxTimerBuff,(TickType_t) 0);

              xTaskNotifyGive(xHandle3);
              printf("Semaphore donné\r\n");
             // printf("i= %d",i);
          vTaskDelay(i);
          i=i+100;
      }
  }
```