Ecole Nationale
Supérieure
de l'Electronique
ENSEA et de ses Applications

Noyaux Temps-réel

Autres moyens de synchronisation et communication

Laurent Fiack
Bureau D212/D060 – laurent.fiack@ensea.fr

Les notifications

Exemple : Notification à la place d'un sémaphore binaire (1/2)

```c
#include "main.h"
#include "usart.h"
#include "tim.h"
#include "gpio.h"
#include "FreeRTOS.h"

uint8_t input_char;

TaskHandle_t h_task_usart1 = NULL;
TaskHandle_t h_task_tim1 = NULL;

/* New character on USART1 */
void USART1_IRQHandler(void)
{
    BaseType_t higher_priority_task_woken = pdFALSE;
    vTaskNotifyGiveFromISR(h_task_usart1,
            &higher_priority_task_woken);

    HAL_UART_Receive_IT(&huart1, &input_char, 1);
    HAL_UART_IRQHandler(&huart1);

    portYIELD_FROM_ISR(higher_priority_task_woken);
}
```

```c
void TIM1_IRQHandler(void)
{
    BaseType_t higher_priority_task_woken = pdFALSE;
    vTaskNotifyGiveFromISR(h_task_tim1,
            &higher_priority_task_woken);

    HAL_TIM_IRQHandler(&htim1);

    portYIELD_FROM_ISR(higher_priority_task_woken);
}

void task_usart1(void * unused)
{
    for(;;) {
        ulTaskNotifyTake(pdTRUE, portMAX_DELAY);
        process_char(input_char);
    }
}

void task_tim1(void * unused)
{
    for(;;) {
        ulTaskNotifyTake(pdTRUE, portMAX_DELAY);
        process_something();
    }
}
```

Exemple : Notification à la place d'un sémaphore binaire (2/2)

```c
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init();

    HAL_UART_Receive_IT(&huart1, &input_char, 1);
    HAL_TIM_Base_Start_IT(&tim1);

    xTaskCreate(task_usart1, "USART1", 256, NULL, 1, &h_task_usart1);
    xTaskCreate(task_tim1, "TIM1", 256, NULL, 2, &h_task_usart1);

    vTaskStartScheduler();
}
```

- 45% plus rapide
- Moins de RAM

Syntaxe : Envoi de notifications

- Envoyer une notification

```
BaseType_t xTaskNotifyGive( TaskHandle_t xTaskToNotify );
```

  - `xTaskToNotify` : Handle de la tâche à notifier
  - Paramètre de retour :
    - Toujours pdTRUE
- Attendre une notification

```
uint32_t ulTaskNotifyTake( BaseType_t xClearCountOnExit,
                           TickType_t xTicksToWait );
```

  - `xClearCountOnExit` : pdTrue pour un sémaphore binaire, pdFalse pour un sémaphore à compte.
  - `xTicksToWait` : Timeout avant de débloquer la tâche.
    - **Attention !** Il n'y a pas eu de notification !
    - Penser à faire de la gestion d'erreur
  - Paramètre de retour : Valeur de la notification avant d'être modifiée

Les queues

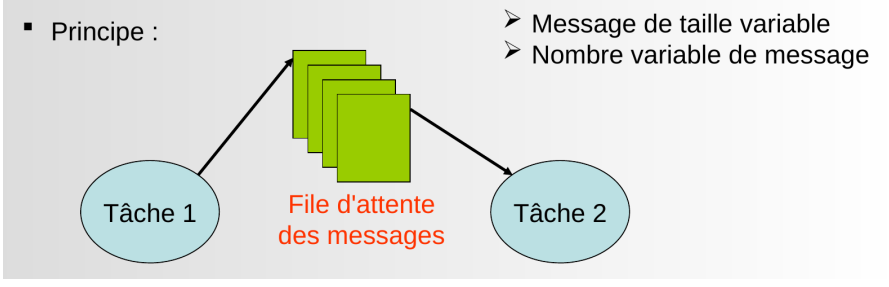
Exemple : Envoi de messages depuis une interruption

```c
#include "main.h"
#include "usart.h"
#include "tim.h"
#include "gpio.h"
#include "FreeRTOS.h"

#define Q_UART1_LENGTH   10
#define Q_UART1_SIZE     sizeof(uint8_t)

uint8_t input_char;

TaskHandle_t h_task_usart1 = NULL;
QueueHandle_t q_usart1 = NULL;

/* New character on USART1 */
void USART1_IRQHandler(void)
{
    BaseType_t higher_priority_task_woken = pdFALSE;

    xQueueSendFromISR(q_usart1, (void *)&input_char,
            &higher_priority_task_woken);

    HAL_UART_Receive_IT(&huart1, &input_char, 1);
    HAL_UART_IRQHandler(&huart1);

    portYIELD_FROM_ISR(higher_priority_task_woken);
}
```

```c
void task_usart1(void * unused)
{
    uint8_t char_to_process;
    HAL_UART_Receive_IT(&huart1, &input_char, 1);

    for(;;)
    {
        xQueueReceive(q_usart1, (void *)&char_to_process,
                portMAX_DELAY);

        process_char(char_to_process);
    }
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init();

    q_usart1 = xQueueCreate(Q_UART1_LENGTH, Q_UART1_SIZE);
    xTaskCreate(task_usart1, "USART1", 256, NULL, 1,
                &h_task_usart1);

    vTaskStartScheduler();
}
```

Les queues

- Aussi appelé Boîtes au lettres ou *Mailbox*
- Communication entre tâches par echange de messages
- Il peut y avoir plusieurs messages sauvegardés dans la queue
- Une queue peut bloquer dans deux conditions :
  - Lecture dans une queue vide,
  - Écriture dans une queue pleine.



Syntaxe : Création d'une queue

- Créer une queue

```
QueueHandle_t xQueueCreate (UBaseType_t uxQueueLength, UBaseType_t uxItemSize);
```

  - uxQueueLength : Nombre de message que peut contenir la queue
  - uxItemSize : Taille (en octets) d'un message
  - Paramètre de retour :
    - Handle de la queue ou NULL en cas d'erreur.

- Détruire une queue

```
void xQueueDelete (QueueHandle_t xQueue);
```

  - xQueue : Handle de la queue à détruire

Syntaxe : Envoi d'un message

- Envoi d'un message dans une queue

```
BaseType_t xQueueSend(QueueHandle_t xQueue, const void * pvItemToQueue,
        TickType_t xTicksToWait);
```

  - `xQueue` : Handle de la queue dans laquelle écrire
  - `pvItemToQueue` : Pointeur sur l'item à envoyer
  - `xTicksToWait` : Timeout avant de débloquer la tâche.
  - Paramètre de retour :
    - `pdTRUE` ou `errQUEUE_FULL` si la queue est pleine
- Lors de l'envoi d'un message, il y a une recopie du contenu du pointeur
- Attention à ne pas envoyer de trop gros objets
- On peut utiliser un pointeur de pointeur

Syntaxe : Réception d'un message

- Reception d'un message depuis une queue

```
BaseType_t xQueueReceive(QueueHandle_t xQueue, void *pvBuffer, TickType_t xTicksToWait);
```

  - `xQueue` : Handle de la queue dans laquelle lire
  - `pvBuffer` : Pointeur sur le buffer de réception
  - `xTicksToWait` : Timeout avant de débloquer la tâche.
  - Paramètre de retour :
    - `pdTRUE` ou `pdFALSE` si la queue est vide
- Lors de la réception d'un message, il y a une recopie du contenu pointé
- Ça fait donc 2 recopies (Envoi + Réception)

Exemple : Envoi d'une grosse structure

```c
#include "main.h"
#include "usart.h"
#include "tim.h"
#include "gpio.h"
#include "FreeRTOS.h"

#define Q_UART1_LENGTH 10
#define Q_UART1_SIZE sizeof(big_struct_t)

TaskHandle_t h_task_send = NULL;

TaskHandle_t h_task_receive = NULL;
QueueHandle_t queue = NULL;

void task_send(void * unused)
{
    big_struct_t big_struct;

    for(;;)
    {
        big_struct = init_big_struct();
        xQueueSend(queue, (void *)&big_struct, portMAX_DELAY);
        // Recopie l'ensemble de la structure dans la queue
    }
}
```

```c
void task_receive(void * unused)
{
    big_struct_t big_struct;

    for(;;)
    {
        xQueueReceive(queue, (void *)&big_struct, portMAX_DELAY);
        // Recopie l'ensemble de la structure depuis la queue
        do_stuff_with_big_struct(big_struct);
    }
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();

    queue = xQueueCreate(Q_UART1_LENGTH, Q_UART1_SIZE);
    // Alloue 10x la taille de big_struct_t
    xTaskCreate(task_receive, "Receive", 256, NULL, 1,
            &h_task_receive);
    xTaskCreate(task_send, "Receive", 256, NULL, 1,
            &h_task_send);

    vTaskStartScheduler();
}
```

Exemple : Passage par pointeur

```c
#include "main.h"
#include "usart.h"
#include "tim.h"
#include "gpio.h"
#include "FreeRTOS.h"

#define Q_UART1_LENGTH 10
#define Q_UART1_SIZE sizeof(big_struct_t *)

TaskHandle_t h_task_send = NULL;

TaskHandle_t h_task_receive = NULL;
QueueHandle_t queue = NULL;

void task_send(void * unused)
{
    big_struct_t big_struct;
    big_struct_t * p_big_struct = &big_struct;

    for(;;)
    {
        big_struct = init_big_struct();
        xQueueSend(queue, (void *)&p_big_struct, portMAX_DELAY);
        // Recopie du pointeur seulement
    }
}
```

```c
void task_receive(void * unused)
{
    big_struct_t * p_big_struct;

    for(;;)
    {
        xQueueReceive(queue, (void *)&p_big_struct,
                portMAX_DELAY);
        // Recopie du pointeur seulement.
        // Attention! Où pointe p_big_struct?
        do_stuff_with_big_struct(*p_big_struct);
    }
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();

    queue = xQueueCreate(Q_UART1_LENGTH, Q_UART1_SIZE);
    // Alloue 10x la taille du pointeur. Quelle taille ça fait?
    xTaskCreate(task_receive, "Receive", 256, NULL, 1,
            &h_task_receive);
    xTaskCreate(task_send, "Receive", 256, NULL, 1,
            &h_task_send);

    vTaskStartScheduler();
}
```

Syntaxe : divers

- Connaître le nombre de messages en attente

```
UBaseType_t uxQueuesMessagesWaiting (QueueHandle_t xQueue)
```

- Connaître la place disponible dans la queue

```
UBaseType_t uxQueueSpacesAvailable (QueueHandle_t xQueue)
```

- Envoi d'un message prioritaire

```
BaseType_t xQueueSendToFront (QueueHandle_t xQueue, const void * pvItemToQueue,
        TickType_t xTicksToWait)
```

- Écraser un élément dans la queue (si la queue a une taille de 1)

```
BaseType_t xQueueOverwrite(QueueHandle_t xQueue, const void * pvItemToQueue);
```

- Lire un message sans le retirer de la liste

```
BaseType_t xQueuePeek(QueueHandle_t xQueue, void *pvBuffer, TickType_t xTicksToWait);
```

Les software timers

Les software timers

- Un software timer permet d'exécuter une fonction après une certaine durée
- On appelle cette fonction une fonction *callback*
- Un timer peut être *one-shot* ou *auto-reload*
- Les software timers utilisent une tâche *daemon* et une queue
  - Toutes les deux masquées à l'utilisateur
- Les fonctions *callback* s'exécutent dans le contexte de cette tâche.
  - Il ne faut pas la bloquer !
- Comme tous les objets FreeRTOS, il faut le **créer** avant de pouvoir l'utiliser

Exemple : Utilisation d'un software timer

```c
#include "main.h"
#include "FreeRTOS.h"

#define PERIOD_TIM1 8
#define PERIOD_TIM2 10

void callback1(TimerHandle_t xTimer)
{
    process1(); // dure 4ms
}

void callback2(TimerHandle_t xTimer)
{
    process2(); // dure 2ms
}
```

```c
int main (void)
{
    HAL_Init();
    SystemClock_Config();

    TimerHandle_t h_soft_tim1 = NULL;
    TimerHandle_t h_soft_tim2 = NULL;

    h_soft_tim1 = xTimerCreate("SWTIM1", PERIOD_TIM1, pdTRUE, NULL,
            callback1);
    h_soft_tim2 = xTimerCreate("SWTIM2", PERIOD_TIM2, pdTRUE, NULL,
            callback2);

    BaseType_t ret;

    ret = xTimerStart(h_soft_tim1, 0);
    configASSERT(pdPASS == ret);

    ret = xTimerStart(h_soft_tim2, 0);
    configASSERT(pdPASS == ret);

    vTaskStartScheduler();
}
```

Syntaxe : Création d'un timer

- Créer un timer

```
TimerHandle_t xTimerCreate (const char * const pcTimerName,
        const TickType_t xTimerPeriod, const UBaseType_t uxAutoReload,
        void * const pvTimerID, TimerCallbackFunction_t pxCallbackFunction);
```

  - `pcTimerName` : Nom pour le debug
  - `xTimerPeriod` : Période du timer en tick. Utiliser `pdMS_TO_TICKS()` pour spécifier le temps en millisecondes.
  - `uxAutoReload` : `pdTRUE` pour activer l'*auto-reload*. `pdFALSE` pour le mode *one-shot*.
  - `pvTimerID` : Donne un identifiant au timer (si un callback est utilisé pour plusieurs timers)
  - `pxCallbackFunction` : Fonction appelée par le timer. Doit suivre le prototype suivant : `void vCallbackFunction (TimerHandle_t xTimer);`
  - Paramètre de retour :
    - Handle du timer ou NULL en cas d'erreur.

Syntaxe : Démarrage et arrêt d'un timer

- Démarrer un timer

```
BaseType_t xTimerStart(TimerHandle_t xTimer, TickType_t xBlockTime);
```

  - `xTimer` : Handle du timer à démarrer
  - `xBlockTime` : La commande passe par une queue. Timeout pour l'éventuel blocage de la queue
  - Paramètre de retour :
    - `pdPASS` si tout s'est bien passé, `pdFAIL` en cas de timeout.

- Arrêter un timer

```
BaseType_t xTimerStop(TimerHandle_t xTimer, TickType_t xBlockTime);
```

  - `xTimer` : Handle du timer à arrêter
  - `xBlockTime` : La commande passe par une queue. Timeout pour l'éventuel blocage de la queue
  - Paramètre de retour :
    - `pdPASS` si tout s'est bien passé, `pdFAIL` en cas de timeout.

Syntaxe : Autres primitives

- Changer la période d'un timer

```
BaseType_t xTimerChangePeriod(TimerHandle_t xTimer, TickType_t xNewPeriod,
       TickType_t xBlockTime);
```

  - xTimer : Handle du timer à démarrer
  - xNewPeriod : Période du timer
  - xBlockTime : Timeout pour l'éventuel blocage de la queue
  - Paramètre de retour :
    - pdPASS si tout s'est bien passé, pdFAIL en cas de timeout.

- Mettre à zéro le timer

```
BaseType_t xTimerReset(TimerHandle_t xTimer, TickType_t xBlockTime);
```

  - xTimer : Handle du timer à démarrer
  - xBlockTime : Timeout pour l'éventuel blocage de la queue
  - Paramètre de retour :
    - pdPASS si tout s'est bien passé, pdFAIL en cas de timeout.

TD

- Reprendre la question 7 du TD3 avec un software timer.