

---

# Programmation en C des microcontrôleurs, Exercices Sujet0ter\_\_c\_\_micro

---

Licence EPOCS \*(ECUE23)

Pierre Pardo

2020-2021

Ce sujet contient 1 exercice(s).



## Comment répondre ?

Ce sujet comporte des exercices, les réponses seront données sous formes de fichier sources (\*.c et/ou \*.h) ou de fichier texte (\*.txt, \*.odt) quand il y a des réponses associées soit:

- Sur un support numérique (clef USB) à la fin du cours ou du TP.
- Par email à [pierre.pardo@univ-amu.fr](mailto:pierre.pardo@univ-amu.fr), au maximum le lendemain du cours ou du TP.

Il y aura souvent plusieurs fichiers aux noms identiques, il faudra transférer les arborescences complètes des fichiers. Le plus simple est de compresser les dossiers.

---

\*Licence professionnelle Électronique Pour Objets Connectés et Smart-Grids



## Carte Nucleo / USB

I Nous allons utiliser la carte Nucleo seule.

Ce projet un peu plus ambitieux implémente un **Gatekeeper**.

L'idée est de ne pas utiliser de Mutex sur une ressource, en créant une tâche qui est la seule à pouvoir y accéder.

Une queue de communication va permettre la synchronisation.

- (a) — Importer le projet **demoGatekeeper** dans **STM32CubeIDE**  
— Renommer le immédiatement en **demoGatekeeper\_imp**.
- (b) Analyser le code source (le **main.c**) pour voir comment est organisé ce programme.  
Ce programme utilise la méthode des rendez-vous (avec un **EventGroupHandle\_t**).  
Le code de synchro est au début de chaque tâche.
- (c) Le fichier **main.c** contient une structure de données :

```
#define MAX_BUFFER_LENGTH 128
#define BUFFER_COUNT 6

enum {SOURCE_NONE, SOURCE_IRQ, SOURCE_DEBUG, SOURCE_COUNT};
typedef struct
{
    uint8_t uSource;
    char cBuffer[MAX_BUFFER_LENGTH];
} sMessage_t;
```

C'est la structure qu'il faudra utiliser pour communiquer avec la tâche **GateKeeper**, qui sera bloquée en attente d'éléments sur une queue de communication.

La macro suivante est un exemple d'envoi de données vers une queue de communication.

```
#define MY_PRINTF(...) do{\
    sMessage_t sb;\
    sb.uSource = SOURCE_DEBUG;\
    snprintf(sb.cBuffer, MAX_BUFFER_LENGTH-1, __VA_ARGS__);\
    xQueueSend(xQueueDebugOut,&sb, pdMS_TO_TICKS(10));\
}\
while(0)
```

- Compléter la tâche **vTaskConfiguration** qui doit configurer une queue de communication.
- Compléter la tâche **vTaskGateKeeper** qui doit être bloquée en attente d'éléments.
  - Attention, obligatoirement sous forme de boucle infinie
- Vérifier la création de la tâche **vTaskGateKeeper** dans le **main**

### Rappel API Queue

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength, UBaseType_t
    ↪ uxItemSize );
BaseType_t xQueueSend( QueueHandle_t xQueue, const void * pvItemToQueue,
    ↪ TickType_t xTicksToWait );
BaseType_t xQueueReceive( QueueHandle_t xQueue, void *pvBuffer, TickType_t
    ↪ xTicksToWait );
```

Compiler et transférer ce programme sur la carte Nucleo.

Lancer un terminal (Putty ou Termite) (115200,8,N,1 sans contrôle).

- Les leds doivent clignoter
- Des messages doivent apparaître régulièrement (tâche **Alive**)
- Un appui sur le bouton bleu doit changer le mode de clignotement, et afficher un message

Les messages renvoyés par la carte sont sur des canaux différents (identifiés par des caractères en début de ligne) :