

## Système embarqué &amp; Programmation RTOS

## ■■■ Programmation RTOS

1 – Soit l'application suivante :

a. Est-ce qu'elle fonctionne correctement ?

*Non, il y a une « race condition », c-à-d un conflit d'accès à la mémoire partagée entre les tâches, c-à-d la variable globale `shared_var`.*

b. Comment la corriger ?

*Il faut une protection par « exclusion mutuelle » pour l'accès à cette variable partagée.*

c. Donnez le code de la correction.

```
static SemaphoreHandle_t mutex;

static int shared_var = 0;

void incTask(void *parameters) {
    int local_var;

    while (1) {
        // Prendre la Sémaphore en mode bloquant
        xSemaphoreTake(mutex, portMAX_DELAY);
        shared_var++;
        Serial.println(shared_var);
        // Libérer la Sémaphore
        xSemaphoreGive(mutex);
        vTaskDelay(random(100, 500) / portTICK_PERIOD_MS);
    }
}

void setup() {
    randomSeed(analogRead(0));
    Serial.begin(115200);
    mutex = xSemaphoreCreateMutex();
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    xTaskCreate(incTask, "Task 1", 1024, NULL, 1, NULL);
    xTaskCreate(incTask, "Task 2", 1024, NULL, 1, NULL);
    vTaskDelete(NULL);
}

void loop() {
}
```

2 – Soit l'application FreeRTOS suivante :

a. À quoi sert le `volatile` ?

*À empêcher que le compilateur optimise la variable, en indiquant qu'elle peut être modifiée par plus d'une thread/tâche.*

b. Décrivez son fonctionnement.

*Alternativement, deux tâches mettent alternativement dans un buffer commun :*

◇ *une chaîne avec son numéro et un compteur personnel ;*

◇ *attends l'une 1s et l'autre 0,5s avant de recommencer après avoir incrémenté son compteur.*

*Le buffer commun possède un rang qui permet de ranger les messages l'un après l'autre jusqu'à une limite.*

*Si la limite est atteinte le message est tronqué.*

*Une troisième tâche tourne en permanence pour détecter l'atteinte de la limite du buffer pour imprimer le message.*

c. Est-ce qu'il est **correct** ?

Non, il peut y avoir des conflits d'accès au buffer et il n'y a pas de synchro pour déclencher correctement l'affichage du buffer.

d. Est-ce qu'il est efficace ?

Non la tâche d'impression est en permanence en accès sur la variable `rang_buffer` pour réaliser la sortie.

e. Proposez une **meilleure version** utilisant les outils disponibles dans FreeRTOS.

```
#define TAILLE_MESSAGE 32

static const uint8_t msg_queue_len = 5;

static QueueHandle_t msg_queue;

void printMessages(void *parameters) {
    char buffer_message[32];

    while (1) {
        xQueueReceive(msg_queue, (void *)buffer_message, portMAX_DELAY);
        Serial.println(buffer_message);
    }
}

void sendMessage(void *paramaters) {
    int nb_messages = 0;
    char buffer_tache[32];

    while(1){
        sprintf(buffer_tache, "[Depuis 1 message %d]", ++nb_messages);
        xQueueSend(msg_queue, (void *)buffer_tache, portMAX_DELAY);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}

void sendMessage2(void *paramaters) {
    int nb_messages = 0;
    char buffer_tache[32];

    while(1){
        sprintf(buffer_tache, "[Depuis 2 message %d]", ++nb_messages);
        xQueueSend(msg_queue, (void *)buffer_tache, portMAX_DELAY);
        vTaskDelay(500 / portTICK_PERIOD_MS);
    }
}

void setup() {

    // Configure Serial
    Serial.begin(115200);

    // Wait a moment to start (so we don't miss Serial output)
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    Serial.println();
    Serial.println("---FreeRTOS Queue Demo---");

    // Create queue
    msg_queue = xQueueCreate(msg_queue_len, TAILLE_MESSAGE*sizeof(char));

    // Start print task
    xTaskCreate(sendMessage, "Send Message", 2048, NULL, 1, NULL);
    xTaskCreate(sendMessage2, "Send Message", 2048, NULL, 1, NULL);
    xTaskCreate(printMessages, "Print Messages", 1024, NULL, 1, NULL);
}

void loop() {
}
```