TP freeRTOS

Thomas Bagrel *Florian Vogt*

OBJECTIF

Développement d'une application temps réel, multitâche mettant en oeuvre le protocole de communication Producteur-Consommateur

Etape 1:

Etape 1:

Créer deux tâches périodiques qui réalisent le modèle Producteur Unique – Consommateur Unique :

- la tâche 1 a une période de 125ms produit des messages « ON » (1) et « OFF » (2) ;
- la tâche 2 a une période de 250ms, elle consomme les messages pour allumer ou éteindre une LED ;
- le buffer de communication est de 10 messages (entiers).

1. Implémenter et tester les tâches en utilisant ***vTaskDelay*** et en affichant les messages.
2. Au bout de combien de temps les deux tâches se synchronisent-elles ?
3. Tester le fonctionnement en affectant la même priorité aux deux tâches, puis en donnant la priorité au producteur, puis au consommateur. Que conclure ?
4. Modifier votre programme pour que les 2 tâches s'exécutent sur le même cœur (***xTaskCreatePinnedToCore***) et retester l'influence des priorités.
5. Modifier le consommateur pour qu'il allume une LED connectée sur le GPIO18.
6. *Facultatif* : utiliser les timers softs (***xTimerCreate***) pour mettre en œuvre les tâches périodiques (ou utiliser ***vTaskDelayUntil***).

1. Les deux tâches se synchronisent une fois que le buffer est complet de manière "stable". C'est à dire, après l'envoi de 19 messages (puisque le producteur a eu le temps de produire deux messages par message lu).
2. Le changement se voit seulement au lancement du programme ; si le producteur a la plus grande priorité, il commence par envoyer 2 messages avant que le consommateur en lise 1. Sinon, si c'est le consommateur qui a la plus grande priorité, au lancement du programme, le producteur produit un message, le consommateur lit ce message, et ensuite le cycle 2 émissions/1 lecture se produit jusqu'à synchronisation (au bout d'une dizaine-vingtaine de messages émis).
3. Lors de nos tests précédents, les 2 tâches s'exécutaient très souvent sur le même coeur (sans qu'on ait besoin d'utiliser xTaskCreatePinnedToCore). Le résultat est donc le

même que les réponses 2 et 3 quand on ajoute cette instruction.

4. 5. 6. Validé avec vous

Etape 2:

Etape 2:

Créer une 2ème tâche consommateur pour implémenter le modèle Producteur Unique – Consommateurs Multiples :

- la tâche conso1 a une période de 250ms et pilote une LED verte sur le GPIO18 ;
- la tâche conso2 a une période de 250ms et pilote une LED rouge sur le GPIO19.

1. Tester le fonctionnement en affectant la même priorité aux deux tâches fonctionnant sur le même cœur (**supprimer les délais**).
2. En donnant la priorité au producteur.
3. En donnant la priorité aux consommateurs. Que conclure ?

2. Lorsque l'on donne la priorité au producteur (sans délai), celui-ci remplit le buffer, puis veille à le garder toujours plein. Il y a donc 10 actions du producteur, puis un enchaînement 1 consommation de message pour la led verte / 1 consommation pour la led rouge. Comme les messages produits sont dans l'ordre ON/OFF/ON/OFF…, c'est toujours la LED verte qui reçoit les messages ON (et qui est donc allumée tout le temps) et c'est toujours la LED rouge qui reçoit les messages OFF (et qui est donc éteinte tout le temps) (sauf bien sûr quand le watchdog intervient).

3. Lorsque l'on donne la priorité aux consommateurs, le début change (puisque le producteur ne produit qu'un seul message, qui est ensuite consommé, et ainsi de suite), mais le comportement global est le même : la LED verte est toujours allumée, et la rouge toujours éteinte.

Etape 3:

Etape 3:

Créer un deuxième producteur pour mettre en place le modèle Producteurs Multiples – Consommateurs Multiples :

- la tâche prod1 a une période de 125ms et produit les « ON » ;
- la tâche prod2 a une période de 125ms et produit les « OFF » ;

1. Tester le fonctionnement en affectant la même priorité aux deux tâches, en donnant la priorité aux producteurs, puis aux consommateurs. Que conclure ?
2. Modifier les producteurs pour qu'ils génèrent les messages depuis les GPIO 4 et 5 :
   a. tester d'abord sur niveau : 1 -> ON ou OFF / 0 -> NOP ;
   b. implémenter avec une interruption pour générer les messages sur le font montant d'un BP : 1 impulsion en 125ms -> ON ou OFF / 0 impulsion -> NOP ;
   Utiliser le principe d'interruption différée.

1. Validé avec vous

   Quand on désactive les délais, les messages ON et OFF sont consommés au hasard par les LEDS rouge et vertes. Quand on donne la priorité aux producteurs, ceux-ci remplissent le buffer avant que les consommateurs puissent se servir. Sinon, quand les consommateurs ont la priorité, les messages sont directement consommés par les consommateurs et le buffer n'a pas le temps de se remplir

2.
   a. Validé avec vous.
   b. (Validé avec vous) Cette fois-ci, les boutons poussoirs font office de déclencheur des tâches : Lorsqu'on appuie sur un des deux boutons poussoirs, nos print se déclenchent et donc le producteur génère le message correspondant; L'opération "NO_OP" n'a pu lieu d'être puisque dès qu'un message est généré, il est consommé et que ces messages sont forcément des ON ou des OFF.