École Nationale des Sciences de l'Informatique	
Module : Systèmes d'exploitation temps réel II2 SLE & II2 ST-IoT	Année universitaire: 2020/2021 Semestre II

Lab 3: FreeRTOS (Sémaphores)

Objectifs :

1. Utilisation des sémaphores dans FreeRTOS.
2. Synchronisation des périodes des tâches.

Output :

Dans ce travail à rendre, vous pouvez vous baser sur le travail du Lab 2.

Ce travail peut être fait en binomes (maximum 2). Vous êtes demandés de rendre le fichier main.c nommé avec la convention suivante : **nom1Prénom1_nom2Prénom2_lab4_SLE/STIoT.c**.

Utilisation des sémaphores

Nous voulons avoir un rendu visuel sur les temps d'arrivée des tâches. Utilisant le code du Lab 2, il est difficile de déterminer quand la tâche de priorité inférieure arrive pour la première fois. En effet, la tâche de priorité supérieure peut utiliser le processeur à ce stade. Donc, elle ne peut pas afficher qu'elle commence son exécution. Cela rend difficile de déterminer si un deadline est dépassé.

Pour ceci, nous considérons deux tâches taskA (3000, 300) et taskB (1000, 100). Comme solution, nous allons ajouter une troisième tâche taskACtrl de plus haute priorité pour nous aider. En particulier, cette troisième tâche ne fera que :

- Afficher que taskA peut commencer son exécution (taskA est arrivée)
- indiquer à taskA qu'elle peut s'exécuter (release)

De cette manière, nous pourrons percevoir le temps d'arrivée de taskA, car la tâche plus prioritaire (taskACtrl) s'exécutera si rapidement qu'elle ne ralentira pas taskB. La tâche la plus prioritaire taskACtrl indiquera à taskA qu'elle peut commencer son exécution. Nous pouvons utiliser un sémaphore. Sachez que la libération d'une tâche de forte priorité est une idée assez courante dans les systèmes d'exploitation et est généralement prise en compte dans le contexte d'interruptions.

Ainsi, nous aurons le même principe théorique d'exécution du lab précédant, sauf que nous aurons une tâche supplémentaire de priorité plus forte en cours d'exécution :

- taskA va utiliser 300ms du temps CPU, mais ne sera exécuté que suite à l'ordre de taskACtrl.
- taskB s'exécutera chaque 1000 ms et utilisera 100 ms de temps CPU. Selon RMS, cette tâche aura une priorité plus élevée que la tâche taskA.
- taskACtrl aura la priorité la plus élevée. Toutes les 3000 ms, taskACtrl annonce l'arrivée de taskA et la réveille. taskA fera ses 300 ms de "travail", affichera la fin d'exécution et attendra taskACtrl de la réveiller à nouveau. Ainsi, taskA aura une période de 3000 ms, mais cette période est contrôlée par taskACtrl.

La façon avec laquelle taskACtrl réveillera taskA est en utilisant un sémaphore. taskACtrl relache le sémaphore (give) toutes les 3000 ms. taskA attendra que le sémaphore soit relaché. Une fois acquit, taskA effectuera son travail de 300 ms de la CPU, puis attendra de prendre le sémaphore à nouveau.

Question : Vous êtes demandé d'implémenter le système décrit précédemment, et de retourner le fichier.

Vous trouvez dans la page 2 une description des fonctions qui vous seront utiles.

Pour utiliser les sémaphores dans ce contexte, vous utiliserez les trois fonctions suivantes :

- `vSemaphoreCreateBinary (xSemaphoreHandle xS)`
  - C'est une macro qui crée correctement un sémaphore. Si xS n'est pas NULL, le sémaphore a été créé avec succès.
- `xSemaphoreTake (xSemaphoreHandle xS, portTickType xBlockTime)`
  - Cette macro tente d'acquérir le sémaphore. S'il n'est pas disponible, il restera jusqu'à ce que le sémaphore soit disponible ou que `xBlockTime` (en ticks) soit écoulé. Il retournera `pdPASS` si le sémaphore a été pris et `pdFALSE` sinon. Si vous ne souhaitez pas vous réveiller sans le sémaphore, mettez une valeur relativement grande dans `xBlockTime`.
- `xSemaphoreGive (xSemaphoreHandle xSemaphore)`
  - libère le sémaphore.