Prof. Guy Bois (Hiver 2020)

INF3610 –Systèmes Embarqués

Hiver 2020

TP No. 1

Groupe 1

1928777 – Mariam Sarwat

1935516 – Louis-Maxime Bois

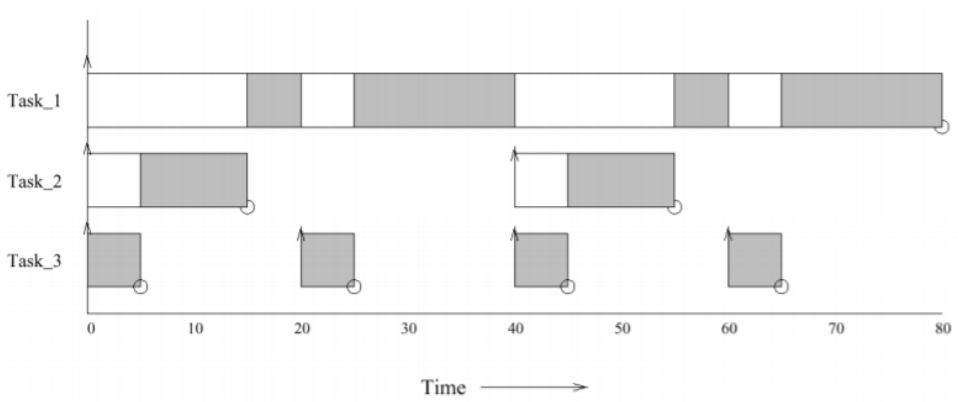
Soumis à : Guy Bois

14-02-2020

Prof. Guy Bois (Hiver 2020)

Exercice 5

En effet, on peut voir avoir le graphique suivant qu'il existe un ordonnancement qui existe. Donc, le théorème de Liu et Layland est une condition suffisante mais pas nécessaire. De plus, ceci est possible de remarquer avec le code de l'exo 5.



1) Indiquez dans quel exercices (2 à 4) la section critique est-elle vraiment utile?

Selon nous, la section critique est vraiment utile dans les exercices 2 et 4 comme en retrouve une possibilité de concurrence entre le robot A et le robot B. En effet, pour ces deux exercices, le robot A donne sa main au robot B avant de finir son exécution. En conséquent, le robot A et B peuvent, par accident, faire des modifications en même temps sur la ressource partagée. Donc, afin d'éviter cette situation, il sera important d'utiliser des mutex qui enveloppe leur section critique respective.

2) Je vous ai indiqué dans le courriel du 5 février d'utiliser (pour exo 2 à 4) les priorités suivantes: Robot_A priorité 60, Robot_B priorité 61, Contrôleur priorité 59. Indiquez si l'assignation suivante fonctionne: Robot_A priorité 60, Robot_B priorité 60, Contrôleur priorité 59. En indiquant si vous obtenez la même trace d'exécution et expliquant comment uC/OS-III gère les égalités.

En changeant la priorité de robot B pour qu'elle soit égale à celle de A, on remarque que la trace d'exécution reste identique. En effet, on perçoit que le contrôleur affiche, en premier, son message de début et, par la suite, les robots A et B affichent à leur tour leurs message de début. Par la suite, comme le contrôleur passe la tâche à faire au robot A à l'aide d'un post, on va toujours passer par le A avant de pouvoir passer au robot B.

Bref, dans le cas d'une égalité des priorités de taches, uC/OS-III exécuterait les tâches dans leur ordre de création. En effet, en inversant l'ordre de création des tâches des robots A et B, on perçoit que l'affichage des messages de début de ces deux robots s'inverse elle aussi (si prio(A) = prio(B)).

3) uC/OS-III offre ce qu'il appelle des Task Sémaphore et Task Message Queues. Expliquez comment ces 2 nouveaux mécanismes auraient pu remplacer les sémaphores de exo2 et les queues de messages de exo4 respectivement (vous n'avez pas à le tester, juste expliquer l'endroit(s) dans le code à modifier). Également, quel avantage apporterait ces 2 nouveaux mécanismes.

Cette nouveauté, soit uC/OS-III, permet aux tâches d'offrir plus de fonctionnalité pour l'utilisation de sémaphores et de queue intégrée. En effet, dans notre cas, on aura pu utiliser les fonctions OSTaskSemPend()/Post() (pour l'exo 2) ou OSTaskQPend()/Post() (pour l'exo 4). Ces fonctions nous permettent d'utiliser les Pend() et Post() sans devoir passer par un objet intermédiaire telle qu'un sémaphore ou une queue. En d'autres mots, l'utilisation de ces fonctions permet à une tâche de recevoir un message, dans le cas des queues, ou un signal, dans le cas des sémaphores, directement du ISR ou d'une autre tâche.

Selon la documentation de Micrium, leur utilisation offre plusieurs avantages comme la simplification du code et une augmentation de l'efficacité du code. De plus, elle pourrait aussi offrir une réduction de la consommation de ressources nécessaire pour le système d'exploitation.

4) Comment feriez-vous dans exo 5 pour indiquer qu'un deadline n'a pas été respecté, par exemple si T1 demande plus de 20 ticks pour une exécution? Expliquez.

En effet, il serait utile d'utiliser une booléenne qui nous permettra à savoir si le deadline a été respecté. En effet, dans l'exo 5 nous avons utilisé la booléenne « *isWorking* » qui sera mise a vrai lorsqu'on commence à travailler dans T1 et après avoir exécuter l'attente actif, on remet cette bool a faux. Finalement, dans T1_Timer afin d'assurer que T1 à terminer son exécution, on vérifie si la bool est à vrai ou faux. Si elle est vrai, on sait alors qu'on a manqué le deadline.

Il serait intéressant de remarquer que l'attente active utiliser permet habituellement que cette demande de temps soit respectée. Dans notre cas, on a que T1 prend 10 ticks pour s'exécuter. Donc si on veut que T1 demande plus de 20 ticks pour son exécution, on remplacerait le « WAITFOR10TICKS » dans le while pour « WAITFORXTICKS » ou X indique le nombre de ticks que T1 prendrait.