

ÉCOLE POLYTECHNIQUE DE MONTREAL

INF3610 : Laboratoire 1

Introduction à μ C-III/OS

29/01/2020

Introduction

Le but de ce laboratoire est de se familiariser avec les différents services mis à votre disposition par $\mu\text{C}/\text{OS-III}$. A la fin de ce laboratoire, vous aurez une vue d'ensemble de comment utiliser un système d'exploitation temps réel. Les connaissances acquises vous serviront aussi lors du laboratoire 2 qui sera plus complexe et applicatif.

Il est pris pour acquis dans ce document que vous êtes familiers avec plusieurs notions de uC expliqués dans le document *INF3610-Lab1_Theorie*. Il est important de le lire avant de commencer le laboratoire!

ATTENTION

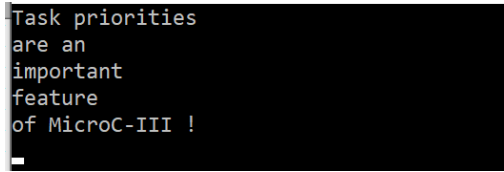
Légende utilisée au cours de ce laboratoire :

- **Le texte en gras** représente des éléments de cours qui doivent être compris pour répondre aux exercices
- Les lignes précédées d'une lettre minuscule (a.) représentent la progression conseillée dans l'exercice
- **Le texte en vert** représente les questions auxquelles vous devrez répondre textuellement dans vos rapports
- **Le texte en rouge** représente des consignes à suivre pour assurer le bon fonctionnement des exercices. **Un non-respect de ces consignes entraînera des pertes de points sévères.**
- Le texte bleu souligné représente des liens vers les ressources disponibles sur Moodle. Il suffit de Ctrl+clic sur ce texte pour y accéder.

Exercices

Exercice 1

Dans ce premier exercice, vous allez devoir utiliser les fonctions de création de tâches pour créer les tâches puis démarrer l'OS. Le but final est d'obtenir cette trace :



```
Task priorities
are an
important
feature
of MicroC-III !
_
```

- Utilisez la fonction [OSInit\(\)](#) avant d'utiliser n'importe quel autre service de μC
- Créez les tâches dans la fonction *main* à l'aide de la fonction [OSTaskCreate\(\)](#). Vous trouverez les informations sur les paramètres à fournir à cette fonction en cliquant sur le nom de la fonction.
- Une fois toutes les tâches créées, utilisez la fonction [OSStart\(\)](#) pour démarrer l'OS
- Faites tourner votre programme une première fois.
- Modifiez le code jusqu'à obtenir la trace attendue

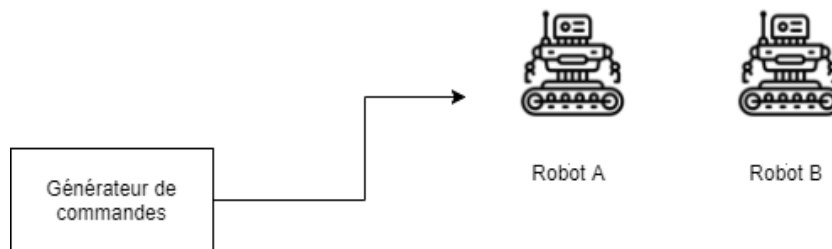
Veuillez respecter les consignes suivantes lors de cet exercice :

- **Ne pas modifier le code des tâches pour cet exercice**
- **Attention! Gérer les cas d'erreurs de manière sommaire (si vous rencontrez une erreur, imprimez un message d'erreur. Pas la peine de faire un message personnalisé en fonction de l'erreur)**
- **Faites attention lorsque vous passez les tableaux d' OS_STK lors de l'appel OSTaskCreate() : vous devez passer l'adresse du début du stack. Or, le stack évolue de l'adresse la plus haute à l'adresse la plus basse : vous devez donc passer la fin des tableaux, pas le début (des explications plus détaillées seront données pendant la période de laboratoire).**
- **Vous pouvez vous inspirer du code de [création d'une tâche](#) sur Moodle dans la section Exemples de codes uC-OSIII**

Exercice 2

Le code des exercices 2, 3 et 4 modélise un système temps réel de deux robots travaillant dans un centre de distribution. La tâche des deux robots est de mettre des objets dans des boîtes. Chaque robot (A et B) est responsable d'objets de types différents.

Les robots reçoivent les commandes de livraison d'un contrôleur qui leur donne des directives à des intervalles de temps aléatoires.



Chaque robot doit exécuter un nombre de commandes (*itemCount*) qui représente le nombre d'objets à déplacer. Ce nombre est généré aléatoirement. Il s'agit en fait d'une valeur aléatoire représentant un nombre de ticks (période d'horloge comme par exemple 1 ms). Donc si le robot A génère 5 aléatoirement, ça veut dire qu'il a 5 objets à transférer et que cela va demander 5 ticks (on assume donc 1 tick/par objet). Sa commande à exécuter va donc durer 5 ticks dans lesquels il devra faire de l'attente active (pour bien comprendre la différence entre attente active et non active référer à l'exemple de *Attente active vs non active* sur le site Web chapitre 2). Il en sera de même pour Robot B.

En ce qui concerne la synchronisation, **vous devez respecter la séquence suivante** : pour chaque nouvelle commande, le contrôleur va d'abord donner la main au robot A (rdv unilatéral)¹. Ce dernier va donc exécuter sa commande (encore une fois, en générant un nombre aléatoire de ticks et en faisant l'attente active), mais avant il va donner la main à Robot B (rdv unilatéral) pour que lui aussi s'exécute avec un nombre de ticks aléatoires. Donc attention, il y a ici possibilité de concurrence entre Robot A et Robot B! Une fois que Robot B a terminé, il va redonner la main à Robot A (rdv unilatéral) puis finalement RobotA va donner la main au contrôleur (rdv unilatéral). On a donc besoin de 4 rdv unilatéral.

Chacun des tapis doit mettre à jour le nombre d'objets déplacés total. Chaque tapis augmentera donc une variable à cette fin. Comme il est important de conserver cette information même en cas de panne de courant, le nombre d'objets est enregistré sur une mémoire externe avec un temps d'accès important, modélisé par les fonctions **readCurrentTotalItemCount()** et **writeCurrentTotalItemCount()** que vous devrez utiliser.

Commentaire de G. Bois : en faisant la solution de exo2 cette semaine, j'ai réalisé que le programme ne fonctionne pas si je donne la priorité maximum au contrôleur. Je dois investiguer davantage (j'ai manqué de temps car je devais voyagé cette semaine). Par contre, si vous donnez à contrôleur la petite priorité, ça fonctionne très bien. Je vous suggère de partir ainsi et si vous avez du temps donnez ensuite au contrôleur la plus grande priorité. Peut être trouverez-vous le problème...On reparlera en classe lundi le 3 février. J'aurai eu le temps d'investiguer...

¹ Attention donner la main veut dire ici envoyer un signal du contrôleur à Robot A, et ce dernier partira si il a suffisamment de priorité.

Veuillez respecter les consignes suivantes lors de cet exercice :

- Ne pas modifier les priorités des tâches pour débiter
- Ne pas modifier directement la variable `total_item_count`.
- Ne pas utiliser plus de 3 sémaphores
- Attention à gérer les cas d'erreurs de manière sommaire (si vous rencontrez une erreur, imprimez un message d'erreur. Pas la peine de faire un message personnalisé en fonction de l'erreur)
- Ne pas utiliser de drapeaux ou de files (vues plus loin)
- Vous ne pouvez pas interrompre le flot du contrôleur, par exemple avec un `OSSemPend()`.

Exercice 3

Le code de l'exercice 3 est une reprise du code de l'exercice 2. Toutefois, il faudra modifier un détail : tous les sémaphores seront remplacés par un seul groupe de [drapeaux d'événements](#) pour minimiser le nombre de synchronisations nécessaires.

Ici vous **devrez toutefois changer la séquence de la manière suivante** : pour chaque nouvelle commande le contrôleur va d'abord donner la main au robot A. Ce dernier va donc exécuter sa commande complètement. Donc, c'est seulement une fois terminé que Robot A va donner la main à Robot B pour que lui aussi exécute sa commande. Comme Robot A a terminé, Robot B va donner la main directement à contrôleur.

Notes :

- Vous pouvez modifier le statut des drapeaux depuis n'importe quelle tâche.
- En cas de problèmes de synchronisation, rappelez-vous que μC est un OS préemptif et donc qu'une tâche plus prioritaire non bloquée s'exécutera toujours avant une tâche moins prioritaire.
- Pour une utilisation des flags voir l'exemple *flag* dans le chapitre 2 du Web.
- Si parfois vous avez un problème [OS FLAG CONSUME](#) de la commande, faites plutôt un [OSFlagPend\(\)](#) suivi d'un [OSFlagPost\(\)](#)...
- Finalement, ici vous devriez pouvoir donner à contrôleur la plus grande priorité.

Veuillez respecter les consignes suivantes lors de cet exercice :

- Ne pas utiliser de queue de messages (files) ou de sémaphores (vous pouvez utiliser des mutex à des fins d'exclusion mutuelle)
- Vous ne pouvez pas interrompre le flot du contrôleur, par exemple avec un `OSSemPend`.

Exercice 4

Le code de l'exercice 4 est une reprise du code de l'exercice 2 et vous utiliserez la même séquence d'exécution que l'exercice 2. Toutefois, notre contrôleur est maintenant capable de fournir des commandes personnalisées. Il est donc responsable de générer aléatoirement les temps d'exécutions des tapis et de les communiquer à Robot A (qui lui pourra les communiquer à Robot B).

De plus vous devez utiliser des [queues de messages](#), c'est-à-dire des files, afin d'acheminer les informations générées dans le contrôleur vers les tâches impliquées.

Veillez respecter les consignes suivantes lors de cet exercice :

- **Ne pas utiliser plus de 3 files (vous pouvez aussi utiliser des mutex ou des sémaphores si approprié (vous devrez donc justifier leur usage)).**
- **Attention à ne pas dupliquer les synchronisations (les files peuvent servir d'éléments de synchronisation dans certains cas)**
- **Vous ne pouvez pas interrompre le flot du contrôleur, par exemple avec un OSSemPend.**
- **N'oubliez pas de libérer la mémoire lorsque nécessaire**

Exercice 5

Ici nous allons changer complètement. Nous allons plutôt nous intéresser au problème d'ordonnancement.

Voici ce qu'on vous demande :

Nous avons vu en classe cette semaine le théorème de Liu and Layland résumé par le power point 8 du cours no 2 :

• *Test d'ordonnancement*

- *Liu and Layland ont démontré que si la condition suivante est rencontrée, on aura toujours un résultat d'ordonnancement:*

$$\sum_{i=1}^N \left(\frac{C_i}{T_i} \right) < N(2^{1/N} - 1)$$

Quand $N \rightarrow \infty$ le terme de droite tend vers $\ln(2) = 69.3\%$

N	Utilization Bound
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.8%

Prenons maintenant l'ordonnancement suivant :

	Period (T_i)	Temps de calcul (C_i)	Priorité (P_i)	Utilisation (U_i)
Task_1	80	40	3	0.5
Task_2	40	10	2	0.25
Task_3	20	5	1	0.25

100% > 78% donc on ne peut rien dire

Par conséquent, faites la démonstration que le théorème de Liu et Layland est une condition suffisante mais pas nécessaire en montrant sous uC OS-III qu'un ordonnancement est possible. Pour le moment, considérez que les temps de changements de contexte et de traitement des interruptions est 0. Aidez-vous du no 32 des exercices et considérez que vous aurez besoin de 3 tâches supplémentaires (qu'on nomme TacheTimer dans le no 32) pour assurer l'ordonnancement. En classe au prochain cours (3 février) nous ferons en exercice le no 32 et je vous donnerai des hints au besoin... En attendant faites un essai

Questions supplémentaires

3 ou 4 questions pour le rapport vont suivre cette semaine...

Barème et rendu

A l'issue de ce laboratoire vous devrez remettre sur Moodle, une fois par groupe de 2, une archive respectant la convention **INF3610Lab1_matricule1_matricule2.zip** contenant :

- Dans un dossier **src**, le code de vos 5 fichiers exo1.c, exo2.c, exo3.c, exo4.c et exo5
- A la racine, un bref rapport contenant les réponses aux questions du laboratoire

Vous devez rendre ce laboratoire au plus tard la veille du prochain laboratoire à minuit (soit 2 semaines après le premier laboratoire)

Barème	
Exécution du code	
Exo1	/2
Exo2	/3
Exo3	/4
Exo4	/4
Exo5	/4
Réponse aux questions	
Question supplémentaire a	/1
Question supplémentaire b	/1
Question supplémentaire c	/1
Question supplémentaire d	/0
Respect des consignes	
Entraîne des points négatifs (peut aussi invalider les points d'un exercice)	
TOTAL	/20

Conclusion

Au cours de ce laboratoire, vous aurez eu l'occasion de vous familiariser avec l'OS temps réel μC . Ce premier laboratoire vous permettra d'être plus à l'aise avec les services fournis par cet OS lors du laboratoire 2, qui sera plus conséquent.