ÉCOLE POLYTECHNIQUE DE MONTREAL

INF3610 : Laboratoire 1

Introduction à µC-II/OS

Guy Bois
Frédéric Fortier
Eva Terriault
1/20/2018

Introduction

Le but de ce laboratoire est de se familiariser avec les différents services mis à votre disposition par µC-II. 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 entrainera 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 :

a. Utilisez la fonction `OSInit()`[1] avant d'utiliser n'importe quel autre service de µC
b. 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.
c. Une fois toutes les tâches créées, utilisez la fonction `OSStart()` pour démarrer l'OS
d. Faites tourner votre programme une première fois.
e. Modifiez le code jusqu'à obtenir la trace attendue

Veuillez respecter les consignes suivantes lors de cet exercice :

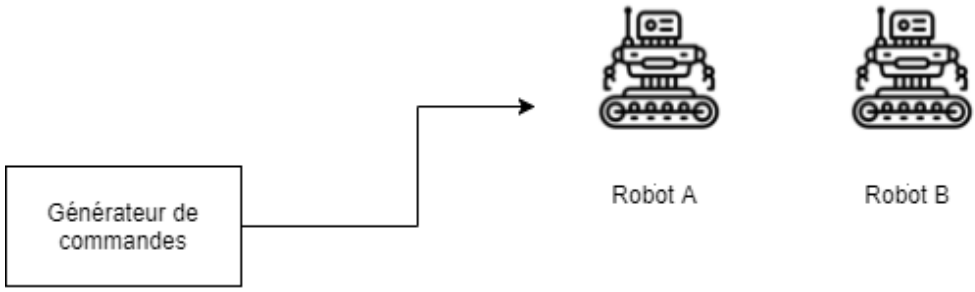
- **Ne pas modifier le code des tâches pour cet exercice**
- **Attention! Gérer les cas d'erreurs de manière sommaire (si vous rencontrez une erreur, imprimez un message d'erreur. Pas la peine de faire un message personnalisé en fonction de l'erreur)**
- **Faites attention lorsque vous passez les tableaux 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).**

[1] Toutes les fonctions µC qui suivent dans ce laboratoire (i.e. commençant par OS) sont clairement définies dans le manuel de référence se trouvant sur le site web du cours (Section 2, sous-section 5, API de µC). Prenez le temps de télécharger le fichier pdf et de le regarder…

Exercice 2

Le code des exercices 2, 3, 4 et 5 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 boites. Chaque robot (A et B) est en charge 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.



Le flot de commande est tel que décrit ci-après :

- Dès qu'une commande est prête, le contrôleur libère une clé sur le sémaphore qu'attend le robot A (créant ainsi un rendez-vous unilatéral). Les robots A et B doivent alors commencer leur travail. La méthode de synchronisation pour le robot B est laissée à votre discrétion.
- Le nombre d'objets à déplacer pour chaque robot est généré de façon aléatoire à chaque commande. Le nombre d'objets n'est pas le même pour les deux robots.
- Le temps requis pour chaque robot afin d'accomplir leur travail correspond au temps requis pour incrémenter un compteur jusqu'à 1000 fois le nombre d'objets à déplacer (attente active).
- Chacun des robots doit mettre à jour le nombre d'objets déplacés total. Chaque robot 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.

a. Créez les tâches controller, robotA et robotB dans le main ainsi que le premier sémaphore sem_controller_to_robot_A.
b. Rajoutez les éléments de synchronisation nécessaires au bon fonctionnement du système au cours des dix commandes du contrôleur.

Veuillez respecter les consignes suivantes lors de cet exercice :

- **Ne pas modifier les priorités des tâches**
- **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().**
- **Pour les exercices 2 à 4, il est accepté que les robots restent simplement bloqués sur lorsque le contrôleur a fini d'émettre des commandes.**

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 (nb. vous pouvez modifier la priorité des tâches si cela vous aide, tant que la tâche contrôleur reste la plus prioritaire).

Notes :

- Une variable comptant la quantité de commandes en attente du contrôleur doit être utilisée pour ne pas manquer de commandes et être partagée entre le contrôleur et le robot A, qui l'incrémentent/décrémentent respectivement, pour ne pas perdre de commandes.
- 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.
- Vous pourriez avoir des problèmes avec l'option OS_FLAG_CONSUME de la commande OSFlagPend()... Il est donc déconseillé de l'utiliser : faites plutôt un OSFlagPend() suivi d'un OSFlagPost().

Veuillez respecter les consignes suivantes lors de cet exercice :

- **Ne pas utiliser de files ou de sémaphores (vous pouvez utiliser des mutex à des fins d'exclusion mutuelle)**
- **Le nombre d'appels à OSFlagPend() (ou similaire comme OCFlagAccept()) est limité à 1 par tâche.**
- **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. Toutefois, notre contrôleur est maintenant capable de fournir des commandes personnalisées. Il est donc en charge de générer aléatoire le temps de préparation des robots et de leur communiquer.

a. Reprenez votre code de l'exercice 2 et utilisez des files afin d'acheminer les informations générées dans le contrôleur vers les tâches impliquées.

Veuillez 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)**
- **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

Le code de l'exercice 5 est une reprise du code de l'exercice 4 . Cependant, une seconde équipe de robots est ajouté au système. Vous devrez modifier les fonctions robotA et robotB (et ajouter les éléments de synchronisation nécessaires) afin que deux instances de chaque tâche roulent en même temps. Vous devez toujours utiliser des files afin de communiquer le nombre d'objets à déplacer à chaque robot à partir du contrôleur.

Il est important que chaque robot communique seulement avec son coéquipier (robot A de l'équipe 1 avec robot B de l'équipe 1, et non robot A de l'équipe 1 avec robot B de l'équipe 2).

L'argument pData de chaque fonction vous sera sûrement utile afin de déterminer dans quelle équipe est quel robot.

Le contrôleur, lui, n'est pas au courant que plusieurs équipes de robots travaillent pour lui. Ainsi, il continue de remplir une seule file de commandes, qui doit être consommée par plusieurs équipes de robots.

Ainsi, il s'agit ici d'une situation courante d'un seul producteur et plusieurs consommateurs. **À l'inverse des exercices précédents, les tâches des robots doivent se suspendre lorsque le contrôleur a fini de produire des commandes (elles ne peuvent restés bloquées).** Vous pouvez vous inspirer du no 4 de l'examen intra hiver 2017 ou du code de la sous-section 4 du chap 2 (intitulé Sémaphore compteur avec plusieurs consommateurs) disponible sur Moodle.

Veuillez respecter les consignes suivantes lors de cet exercice :

- **Commencez par faire en sorte que l'exercice 4 fonctionne en créant deux fois les tâches de chaque robot. Il vous faudra sûrement utiliser des éléments de synchronisation supplémentaires ainsi que utiliser l'argument pData des fonctions.**
- **Ensuite, assurez-vous que les tâches robot A et robot B se terminent correctement lorsque la production de commande est terminée.**
- **Vous pouvez revoir vos méthodes de synchronisation entre tâches au besoin, utiliser plus ou moins de sémaphores, etc.**
- **Aucun code de départ n'est donné pour cet exercice : copiez exo4.c vers exo5.c lorsque vous aurez terminé 4.**

Questions supplémentaires

a. **Dans le contexte du problème d'inversion de priorité et de ses solutions, précisez ce qu'on entend par temps de blocage versus temps de préemption.**
b. **Toujours dans le contexte du problème d'inversion de priorité, donnez 2 avantages d'utiliser le protocole ICPP par rapport à l'héritage de priorité.**
c. **Donnez un exemple de situation où il serait préférable d'utiliser un sémaphore plutôt que des drapeaux d'événements.**
d. **Donnez une courte appréciation du laboratoire (temps consacré, explications, clarté de l'énoncé, difficultés, etc).**

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 4 fichiers exo1.c, exo2.c, exo3.c, exo4a.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	
Entraine 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.