

INF3610

Systèmes embarqués

Laboratoire 1

Introduction à μ C/OS-II

Soumis par

Félix Boulet, #1788287

Giuseppe La Barbera, #1799919

le 21 septembre 2017

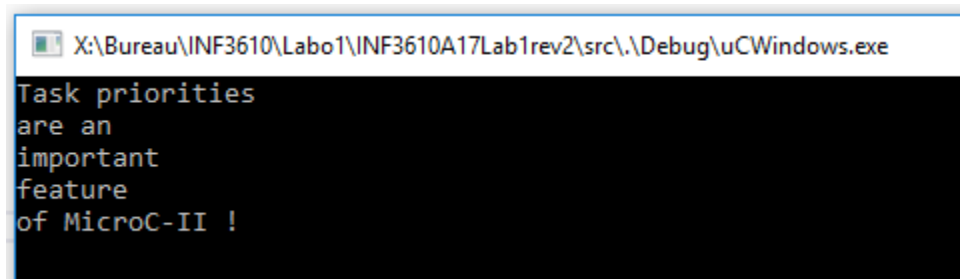
Exercice 1 :

Lors de l'exécution initiale du programme (en se servant des priorités prédéfinies pour chacune des tâches), on obtient la trace suivante :



```
X:\Bureau\INF3610\Labo1\INF3610A17Lab1rev2\src\.\Debug\uCWindows.exe
are an
of MicroC-II !
Task priorities
important
feature
```

Cela est dû au fait que la tâche 1 affiche en fait « are an », et est plus prioritaire que la tâche 2, qui affiche « Task priorities » (PRIO1 = 5 < PRIO2 = 7). Il en va de même pour les autres tâches, et on obtient une trace qui ne correspond pas à celle que l'on désire obtenir. Pour régler ce problème, on redéfinit les priorités de chacune des tâches (PRIO1 = 11, PRIO2 = 10, PRIO3 = 12, PRIO4 = 13, PRIO5 = 14) afin que celles-ci s'exécutent dans l'ordre désiré, avec pour résultat la trace ci-dessous :



```
X:\Bureau\INF3610\Labo1\INF3610A17Lab1rev2\src\.\Debug\uCWindows.exe
Task priorities
are an
important
feature
of MicroC-II !
```

Exercice 5 :

- a) Un exemple de situation où il serait bénéfique d'utiliser un sémaphore plutôt que des drapeaux d'événements serait la situation classique de l'achat de billets sur un site internet. En effet, puisqu'un sémaphore peut ne pas être binaire (il peut avoir une valeur supérieure à 1), il est possible de s'en servir comme une variable globale partagée entre différentes tâches. Ainsi, à chaque achat de billet par exemple, le sémaphore décrémente, jusqu'à atteindre la valeur 0 et empêcher tout achat subséquent. Il serait difficile de reproduire ce comportement avec des drapeaux d'événements sans utiliser une variable globale dédiée manipulée à travers un mutex (pour l'exclusion mutuelle), comme on l'a fait dans l'exercice 3 pour tenir le compte du nombre de commandes restantes. C'est ainsi beaucoup plus commode d'utiliser un sémaphore plutôt que des drapeaux d'événements dans ce genre de situation.

b) Si on pense bêtement au nombre total de commandes que l'on a à effectuer, la taille minimale serait de 10 (telle qu'implémentée dans l'exercice). Toutefois, si on veut optimiser le code (en termes de mémoire utilisée, ce qui peut être crucial dans le cas d'un système embarqué), on peut en arriver à un nombre réduit en utilisant la logique suivante :

- Le contrôleur s'exécute à des intervalles aléatoires de $([0, 8] + 5) * 10$ unités de temps. Ainsi, l'intervalle minimal entre deux exécutions est de 50 unités de temps.
- Le délai accordé pour chaque tâche se trouve également dans un intervalle aléatoire. Le robot A peut avoir un temps de préparation de $([0, 7] + 3) * 10$ unités de temps, le robot B de $([0, 7] + 6) * 10$ unités de temps, et le transport de $([0, 5] + 9) * 10$ unités de temps. Ainsi, le temps maximal d'exécution de chaque tâche, prises séparément, est de 100 u.t. pour le robot A, 130 u.t. pour le robot B, et 140 u.t. pour le transport.
- Le robot A et le robot B se préparent de façon simultanée en théorie (et dans le meilleur des cas). On peut donc utiliser le temps du robot B comme temps maximal de préparation pour les deux robots. Le temps de transport doit ensuite y être ajouté (les robots doivent être prêts avant de pouvoir effectuer le transport). On a donc un total de 270 unités de temps au maximum par commande.
- Puisque dans le pire des cas le contrôleur pourrait émettre à toutes les 50 u.t. une commande qui prend un total de 270 u.t. pour s'effectuer, il faudrait que les files puissent emmagasiner les détails de $\lceil 270/50 \rceil = 6$ commandes. Il serait également possible qu'elles emmagasinent seulement 5 commandes, mais c'est en assumant une exécution parfaite (la première commande est immédiatement consommée, donc après 270 u.t. et 5 commandes supplémentaires produites, une place se libère pour la commande produite à 300 u.t.).
- En réalité, puisque l'exécution est rarement parfaite (les délais ne sont pas garantis), il est plus sécuritaire d'avoir des files légèrement plus grandes.

c) Le laboratoire est approprié pour un premier laboratoire. Il nécessite une bonne application des notions de synchronisation et du cours prérequis INF2610, qui est révisé dans les premières périodes de INF3610. En termes de temps, il n'est ni trop court ni trop long (2 semaines sont amplement suffisantes pour le compléter, mais il ne prend pas une heure seulement non plus), et le niveau de difficulté est approprié pour autant qu'on lise la documentation fournie sur $\mu C/OS-II$. Une suggestion toutefois serait de faire un petit rappel dans les consignes sur où utiliser des pointeurs ou non, car ce n'est pas toujours clair (en particulier dans l'exercice 4, qui nécessite des conversions vers des pointeurs *void*). Avec du C++ un peu rouillé, ce n'est pas nécessairement évident.