HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Yann Thoma

Laboratoire de Programmation Concurrente

semestre printemps 2015 - 2016

Prise en main des threads Qt

Temps à disposition : 2 périodes (travail débutant en semaine 1)

1 Objectifs pédagogiques

— Se familiariser avec l'environnement de programmation Qt ;

— Réaliser son premier programme concurrent en C++ ;

— Tester les problèmes de concurrence.

2 Cahier des charges

Le programme qui vous est donné effectue une simple incrémentation d'un nombre un certain nombre de fois. Cette incrémentation est faite par plusieurs threads, et ce potentiellement en parallèle (ou pseudo-parallèle). Si vous le lancez et le compilez, vous devriez observer des valeurs étranges pour la valeur finale du compteur.

Le dialogue lancé par l'application vous permet de choisir le nombre de threads à lancer et le nombre d'itérations à effectuer. Une pression sur le bouton *Start* lance effectivement les threads et lorsqu'ils ont tous terminé la valeur finale du compteur ainsi que le ratio (valeur finale observée/valeur finale attendue) sont affichées.

A partir du code qui vous est donné tentez quelques expériences pour améliorer la valeur finale observée. Pour ce faire essayez de mettre en place de l'attente active en exploitant des variables partagées par l'ensemble des threads. Utilisez donc des variables de classe (**`static`**). Le fichier à modifier est `mythread.cpp`, où vous pouvez observer l'incrémentation d'un compteur partagé.

Durant vos essais, vous pouvez jeter un oeil à la méthode `QThread::yieldCurrentThread()`. Consultez la documentation de Qt pour savoir ce qu'elle fait. Un autre point à tester potentiellement est l'ajout d'une instruction de barrière mémoire :

```
__asm__ volatile ("sfence":::"memory");
```

Si vous avez installé Qt sous Windows avec Microsoft Visual Studio comme compilateur, utilisez :

```
_ReadWriteBarrier();
```

Ceci permet de synchroniser les mémoires caches, et en le plaçant à des endroits judicieux cela permet de rendre le code plus sûr (bien que moins performant).

Enfin, n'hésitez pas à tester votre code en forçant l'usage d'un seul coeur :

— Sous Linux, en ligne de commande : `taskset 1 ./counter`

— Sous Windows, en ligne de commande :`start /AFFINITY 1 counter.exe`

— Sous MacOS, eh bien, à vous de trouver...

3 Développement Qt

Une fois installé l'environnement Qt, vous avez à disposition l'IDE QtCreator. Il permet de gérer les projets Qt, d'éditer vos programmes, de les débugger, ...

Un projet est décrit dans un fichier .pro. En l'ouvrant l'identification du compilateur à utiliser vous sera demandé. Choisissez-en un pour une version de Qt>=5.2.

La compilation peut se faire en *shadow build* ou non. *shadow build* signifie que tous les fichiers générés le sont dans un dossier séparé des sources. Soyez donc attentif quant à l'emplacement de votre exécutable. Dans QtCreator, l'onglet gauche *Projects* permet de spécifier si l'on veut travailler en *shadow build* ou non. Suivant les laboratoires il se peut que nous vous indiquions s'il faut l'exploiter ou non.

Le moyen de compiler et lancer votre programme depuis QtCreator est laissé à votre perspicacité de programmeur.

4 Travail à rendre

Il s'agit d'une prise en main. Amusez-vous, posez toutes les questions nécessaires, et ne rendez rien.