(R5.05) R5.05 - TP sur les thread - partie 2

Le inconnue durée : 4 heures

Alain Casali alain.casali@univ-amu.fr
Aix Marseille Université
I.U.T. d'Aix en Provence - Département Informatique

Remarques

- Certains propos tenus dans ce TP tenus à but humoristiques, historiques. Merci de ne pas les dévoyer.
- Ajoutez la ligne `LIBS += -pthread` à votre projet (votre ficher `.pro`) sous `QT Creator`.
- On va compiler avec un compilateur intégrant les instructions du C++20, on peut considérer que c'est +/- safe => pas besoin de wrappers :)

Le but de ce TP est tester quelques sous-programmes créant des thread (et non des processus fils), ou en utilisant la bibliothèque OpenMP. et de comprendre leur comportement.

[troll] Comme le disait spiderman : "A grand pouvoir, grande reponsabilité"

Les objectifs de ce TPs, sont :
— de vous donner accès à l'intégralité des ressources d'un OS. Faites attention svp !
— quand vous coder une fonction, faites attention à son implémentation :
  1. dans un environnement mono-processus ;
  2. dans un environnement multi-processus ;
— de faire quelques tests dans un environnement multi-thread ;
— de comprendre les *effets de bords* de cet environnement multi-thread

Exercice 1. *Threader un processus existant*

Sur Ametice, vous trouverez la correction du client / serveur `echo` écrit en C / C++.

Question 1.1 () :

Modifier le code du serveur de façon à ce que chaque client soit servi par un `thread`.

Remarque : Pour simuler un processus client un peu long à répondre, faites endormir le `thread` courant (ie celui du serveur pendant plusieurs secondes).

Exercice 2. *Avant Open-MP*

Soit la fonction `generationOfACollection` suivante :

```
void generationOfACollection (const T & begin,  const T & end,
                              const unsigned & min = 0u,
                              const unsigned max = RAND_MAX);
```

Cette fonction a pour but de générer des valeurs aléatoires dans la tranche `[min, max[` d'une collection (*ie* un tableau (`vector` ou un `array`)). Pour simplifier la suite de ce TP, tous les tableaux auront une de taille de 1E6 éléments. En mode mono-processus / mono-thread, il faut exécuter le code suivant :

```
unsigned nbThread (XXX);
vector<unsigned> vUInt (1E6);
size_t range (vUInt.size() / nbThread);
for (size_t i = 0; i < vUInt.size(); i+=range){
        generationOfACollection (vUInt.begin() + i, vUInt.begin()+i+range, 0, 100);
    }
```

Question 2.1 () :

Encapsuler cet appel de fonction avec `nbThread`. Ainsi, chaque `thread` est chargé de remplir sa propre portion du tableau.

[NB :] n'oubliez pas la boucle `join ()` après cet boucle `for ()`.

Exercice 3. *Avec Open-MP*

Question 3.1 () :

Reprenez l'exercice précédant, et remplacez le lancement des `threads` par une boucle `for` en utilisant les directives Open-MP.

Pour activer les directives `Open-MP` il faut

1. include la bibliothèque `omp.h` ;
2. ajouter la directive de compilation `-fopenmp`.

Exercice 4. *Problème avec Open-MP*

Dans cette partie, on va montrer les limites d'Open-MP (ou plutôt de nos façons de programmer salement !).

Une des boucles de recherche d'un élément dans un tableau pourrait être la suivante :

```
template <class T, class U>
size_t myFind((const T & begin,  const T & end,
               const U & val) {
    for (const auto & it (begin); it < end; ++it){
        if (val == *it) return (it - begin);
    }
    return -1;
}
```

Question 4.1 () :

threader cette boucle avec `Open-MP`.

Question 4.2 () :

monter qu'elle peut renvoyer des résultats faux.

Exercice 5. *Protocole de performance des algo des algorithmes de tri*

On suppose que nous avons à disposition 4 algorithmes de tris (tri par insertion, tri à bulle, tri par sélection / échange, tri fourni par le langage).

Établissez un protocole de test de façon à comparer les temps (moyen, médian, min, max) mis par chaque méthode pour trier le même tableau.

Question 5.1 () :

Ecrire une version avec des `thread` ;

Question 5.2 () :

Ecrire une version avec des `Open-MP`.