(R3.05) R3.05 - TP sur les thread

Le inconnu– durée : 2 heures
Alain Casali alain.casali@univ-amu.fr
Aix Marseille Université
I.U.T. d'Aix en Provence - Département Informatique

Institut Universitaire de Technologie
Aix*Marseille Université
Département Informatique, Aix

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++11, 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), 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. *Quelles sont les perfs d'une machine Von Neman en 2023 ?*

Question 1.1 () :
Comment peut-on connaitre le nombre de coeur dont on dispose ?

Question 1.2 () :
Pourquoi est-ce important de la connaitre ?

Question 1.3 () :
De combien de coeurs disposez-vous ?

Exercice 2. *2 (ou 3) Premiers threads*

Soit la fonction `fct` suivante :

```
void fct (){
    cout << "␣Hello␣World␣!␣" << endl ;
    cout << "Bonjour␣" << flush ;
    cout << "tout␣" << flush << "␣le␣monde" << endl;
}
```

Lancer cette fonction dans 2 thread différents dans le `main ()`, sans faire de point de synchronisation ! Vous devez constater que vous avez le message suivant dans le terminal :

```
terminate called without an active exception
```

Question 2.1 () :
expliquer pourquoi on capture une exception.

Question 2.2 () :
palier à ce problème en utilisant la méthode `join ()`.

Remarque : Après plusieurs exécutions de votre programme, même ce-dernier ne plante plus, Vous deviez constatez plusieurs problèmes d'affichage. Nous allons les corriger dans l'exerce suivant.

Exercice 3. *Et les sections critiques ? ?*

Dans cet exercice, on souhaite limiter l'accès un `thread` au terminal. Pour cela on va :

1. déclarer une variable globale sur le programme :
   ```
   mutex mut;
   ```
2. dans la fonction `fct ()` : faire une opération `.lock ()` sur le `mutex` au début du la fonction ;
3. dans la fonction `fct ()` : faire une opération `. unlock ()` sur le `mutex` à la fin de la fonction ;

Question 3.1 () :
Lancer plusieurs fois votre exécutable. Vous devez constatez des différences dans les affichages. Prenez quelques minutes pour expliquer, pour chaque exécution, quel est le `thread` qui fait les affichages dans le `shell`.

Exercice 4. *Identifiez un thread*

Dans cette partie, on souhaite afficher :

1. l'identifiant du thread courant à la fin de `fct` ;
2. l'identifiant de `t1` dans le *main thread*.

Question 4.1 () :
Faites les modifications nécessaires.

Question 4.2 () :
Il se passe quoi si vous affichez l'identifiant de `t1` après l'instruction

```
t1.join ();
```

Lancer plusieurs fois votre exécutable. Vous devez constatez des différences dans les affichages. Prenez quelques minutes pour expliquer, pour chaque exécution, quel est le `thread` qui fait les affichages dans le `shell`.

Exercice 5. *Et les sections critiques ? ? - V2*

En reprenant un de vos exercices précédant, commenter la ligne

```
mut.unlock();
```

Question 5.1 () :

Lancer plusieurs fois votre exécutable. Vous devez constatez que votre processus ne se termine pas ;

Question 5.2 () :

Palier à ce problème en utilisant un `lock_guard`.

Exercice 6. *Incrémentation d'une variable globale*

Dans cette partie, on souhaite exécuter le code suivant dans plusieurs threads :

```
// variable globale
unsigned i (0);
void inc ()
{
    ++i ;
 }
```

Question 6.1 () :

Lancer cette fonction dans 2/3 thread. Afficher le résultat à la fin de l'appel à tous ces threads. Que constatez vous ?

Question 6.2 () :

Pour faire planter notre assertion, on va devoir lancer plusieurs centaine de threads. Pour cela, il faut :

1. créer un vecteur de thread ;
2. chaque thread lance la fonction `inc ()` ;
3. attendre la fin de tous les threads
4. faire la vérification une fois tous les threads exécutés.

=> faire plusieurs tests avec un nombre de threads conséquent (i.e. >= 2000).

Question 6.3 () :

Palier à ce problème en utilisant un `lock_guard`.

Exercice 7. *Incrémentation d'une variable locale*

Dans cette partie, on souhaite exécuter le code suivant dans plusieurs threads :

```
void inc (unsigned & i )
{
    ++i ;
 }\\
 int main ()
 {
     // variable globale
    unsigned i (0);
    thread th1 (inc,  ref (i));
    ...
}
```

Question 7.1 () :

Lancer cette fonction dans 2/3 thread. Afficher le résultat à la fin de l'appel à tous ces threads. Que constatez vous ?

Question 7.2 () :

Pour faire planter notre assertion, on va devoir lancer plusieurs centaine de threads. Pour cela, il faut :

1. créer un vecteur de thread ;

2. chaque thread lance la fonction `inc ()` ;
3. attendre la fin de tous les threads
4. faire la vérification une fois tous les threads exécutés.

=> faire plusieurs tests avec un nombre de threads conséquent (i.e. >= 2000).

Question 7.3 () :

Palier à ce problème en utilisant un `lock_guard`.

Exercice 8. *Incrémentation d'une variable globale / locale sans mutex*

Dans cette partie, on ne veut pas pas utiliser de `mutex`.

Question 8.1 () :

Comment fait-on pour s'assurer de l'attomicité des opérateurs sur un type "*entier*" ?

Exercice 9. *Initialisation et affiche d'un tableau dans un mode concurentiel*

Écrire une fonction qui initialise un vecteur d'entiers naturels, et une autre qui affiche le contenu d'un vecteur d'entiers naturels. Lancez chaque fonction dans un thread.

Question 9.1 () :

faites en sortent qu'il n'y ait aucun soucis.

soucis.

Exercice 10. *Nos meilleurs amis : les italiens <3*

Reprenez le corrigé du TD sur les philosophes mangeurs de spaghetti.

Question 10.1 () :

Implémentez une des solutions.