SCIENCES SORBONNE UNIVERSITÉ

Examen Réparti 2 PR
Master 1 Informatique Jan 2019

UE 4I400

Année 2018-2019

2 heures – **Tout document papier autorisé**
Tout appareil de communication électronique interdit (téléphones...)

Introduction

- Le barème est sur 22 points et est donné à titre indicatif et susceptible d'être modifié.
- Dans le code C++ demandé vous vous efforcerez d'ecrire du code correct et compilable, mais les petites typos ne seront pas sanctionnées.
- De même les problèmes d'include et de namespace ne sont pas pertinents (pas la peine de qualifier les `std::`, ni de citer tous les `#include`)
- On suppose également que **tous les appels système se passent bien**, on ne demande pas de contrôler les valeurs de retour (donc pas de `perror` ou `errno.h`).

Le sujet est composé de trois exercices indépendants qu'on pourra traiter dans l'ordre qu'on souhaite.

1 Synchronisations (11 points)

Notions testées : concurrence et synchronisations.

On considère une variante du problème du dîner des philosophes, proposé par Dijkstra. On considère N threads, qui travaillent ensemble sur un vecteur tab de N cases.

Chaque thread d'indice i a le comportement suivant (répété 100 fois):

- Il "réflechit" pendant un temps variable, ce qui est représenté par l'invocation à `bool think()`, une fonction qui ne sera pas détaillée dans ce sujet.
- S'il est satisfait du résultat de sa réflexion (`think` a rendu vrai), il échange la valeur de la case $tab[i]$ avec celle de $tab[(i+1)\%N]$. On a donc une gestion circulaire du tableau.

Question 1. (3 points) Complétez la fonction `main` qui crée `tab` (initialisé avec comme contenu les entiers de 0 à $N-1$), engendre les N threads "philosophe", attends leur terminaison et enfin affiche le contenu de tab sur la sortie standard. Donnez la signature de la fonction `philosophe` cohérente avec ce main, et compléter son corps.
NB : on ne demande pas encore de coder de synchronisations.

main.cpp

```cpp
const int N = 5;

// temps de reflexion arbitraire
// rend true ou false de façon arbitraire
// la fonction ne sera pas détaillée.
bool think ();

// fonction executée par chacun des thread
// Signature à fournir en Q1.
???? philosophe ( ???? ) {
      for (int iter = 0 ; iter < 100 ; iter++) {
            if (think()) {
                  // echanger les cases d'indice i et (i+1) % N de tab.
            }
      }
}

// Question 1 : compléter le corps de main.
int main() {
      // declarer tab
      // initialiser tab avec les entiers 0 à N-1
      // engendrer N threads exécutant "philosophe" ; leur passer leur indice de création et
           tab
      // attendre la terminaison de tous les threads
      // afficher tab
      return 0;
}
```

Question 2. (2 points) Le programme actuel, sans synchronisations est évidemment incorrect. Ajoutez des synchronisations pour protéger l'accès aux cases du tableau, tout en **maximisant** la concurrence : deux philosophes qui ne sont pas adjacents doivent pouvoir travailler en même temps. Vous expliquerez comment déclarer ces éléments (dans le main) et vous donnerez les modifications à réaliser dans la fonction `philosophe`.

Question 3. (1 point) Le programme actuellement se bloque parfois avant de se terminer, expliquez pourquoi en exhibant une séquence d'exécution possible du programme.

Pour remédier à ce problème, vous allez coder un système de tickets de manière à ce que au plus $N-1$ philosophes puissent commencer à travailler en même temps. Chaque philosophe prend un ticket avant de commencer à travailler, et le rend avant d'entamer une nouvelle étape de réflexion. Dans les questions suivantes, on demande de développer une classe `TicketBarrier`, son constructeur prenant un entier (le nombre de tickets disponibles), ses opérations `void enter()` et `void leave()`.

Question 4. (1 point) Donnez les modifications à apporter au code du programme pour utiliser cette classe `TicketBarrier`. La classe elle-même sera implantée dans les questions suivantes.

Question 5. (2 points) Ecrire une version de la classe `TicketBarrier` en n'utilisant que les primitives fournies dans `std::` par C++11.

Question 6. (2 points) Ecrire une version de la classe `TicketBarrier` en n'utilisant que des primitives POSIX.

2 Invocation Distante (8 points)

On considère l'interface `IValue` et sa réalisation triviale `Value` suivante :

IValue.h

```
#pragma once

class IValue {
public :
      virtual void set (int val) = 0;
      virtual int get () const = 0;
      virtual ~IValue() {}
};

class Value : public IValue {
      int val;
public :
      Value (int v=0):val(v) {}
      void set (int v) { val = v; }
      int get () const { return val; }
      ~Value() {}
};
```

Notre objectif est de permettre un accès par un client à une instance de `Value` distante, c'est à dire hebergée sur une autre machine serveur, en suivant les principes d'un proxy distant. On considère que les machines sont connectées par un réseau IPv4 classique. L'exercice est centré sur le réseau, et ne nécessite donc pas de mécanismes liés à la concurrence.

Question 1. (1 point) Selon que l'utilisateur du logiciel passe les addresses de machine sous la forme "`hote.reseau.domaine`" (e.g. "`www.google.fr`") ou sous la forme d'une ip en quatre parties "`a.b.c.d`" (e.g. "192.168.0.12"), quelle primitive(s) de conversion faut-il utiliser pour obtenir une addresse IP au format machine (utilisable dans un `struct sockaddr_in` par exemple) ? Expliquez rapidement la différence entre ces familles de fonctions.

On suppose dans les questions suivantes que l'on dispose d'une fonction `in_addr_t getIP (const string & host)` implantant ce service de conversion.

Question 2. (2,5 points) Ecrivez une classe `ValueServer` qui sera hebergée sur la machine serveur et stocke une référence vers une instance de `IValue` (le sujet). Elle est munie d'un constructeur dont vous préciserez la signature, et d'une opération `void treatClient()`. A la construction, la classe doit mettre en place les mécanismes réseau utiles pour pouvoir recevoir des demandes de clients. La méthode `treatClient` traite une demande d'un client : établissement de la connection, reconnaitre si le client veut invoquer "set" ou "get", traiter sa demande **par délégation sur le sujet** et enfin mettre fin à la connection.

Question 3. (2,5 points) Ecrivez une classe `ValueProxy` qui sera hebergée sur la machine client. Elle est munie d'un constructeur dont vous préciserez la signature, et réalise l'interface `IValue` en appui sur une connection réseau vers un `ValueServer`.

Question 4. (2 points) Ecrivez un main client et un main serveur qui utilisent ces classes.

3 Questions de Cours (3 points)

Question 1. (1 point) Expliquez dans quel contexte la primitive `select` peut être utile.

Question 2. (1 point) On considère un programme où deux threads se synchronisent à l'aide de signaux (primitives `kill` et `sigsuspend`). Le comportement obtenu ne correspond pas aux attentes du programmeur. Expliquez pourquoi.

Question 3. (1 point) Dans quelle(s) situation(s) doit-on utiliser un segment de mémoire partagée **nommé** (au lieu d'un segment anonyme) ?