TP - ASR7 Programmation Concurrente

—

Synchro !

Sylvain Brandel, Guillaume Damiand, Laurent Lefèvre, Matthieu Moy, Grégoire Pichon

Printemps 2022

Avec l'aide du travail que vous avez fourni jusqu'à maintenant, nous allons ici nous intéresser à la mise en place de synchronicité dans l'exécution de tâches.

I Synchronisation simple

Le but ici est d'écrire un programme qui effectue les tâches A_1 et A_2 en parallèle, puis B. Les tâches A_1 et A_2 exécutent du code arbitraire, par exemple Fibonacci() sur un nombre tiré aléatoirement (ou tout simplement `sleep`;D). La tâche B attend que les tâches A_1 et A_2 soient terminées avant de s'exécuter. Attention, les trois tâches doivent s'exécuter en parallèle : vous ne pouvez pas simplement lancer , A_1 et A_2 en parallèle, puis faire un `join` avant de lancer B ! Voici le squelette du `main` de votre programme :

```
1  int main() {
2    for(...) {
3      std::thread A_1(...);
4      std::thread A_2(...);
5      std::thread B(...);
6      A_1.join(); A_2.join(); B.join();
7      }
8    return 0;
9  }
```

— Proposez une solution et discutez-en avec votre encadrant.

— La solution est-elle facilement généralisable au cas n threads ?

II Barrière de synchronisation

— À quel type de problème correspondait l'énoncé précédent ?

Ceux qui feront l'UE Parallélisme l'année prochaine verront des couplages de codes MPI/OpenMP : il existe des primitives qui peuvent être utilisées pour simplifier la programmation de programmes multi-processus (la gestion des communications à travers les sockets se voit comme "Envoyer cette structure de données au processus X" alors que le processus X fait un "Réception d'une structure de données" : pas besoin des `listen()` ou `fork()` vus en Système d'Exploitation en L2 !) et multi-thread (on laisse le compilateur gérer nos demandes de création

de threads exprimées par des `#pragma` ! Par contre, il faut bien comprendre les opérations de réduction [1]).

II.1 Travail à réaliser : la barrière

Nous vous proposons ici de coder une barrière de synchronisation afin de faire fonctionner un code jouet. Le code principal, dans votre `main`, doit demander à l'utilisateur le nombre de threads qui s'exécuteront, ainsi qu'une valeur `nbtours` qui représente le nombre d'itérations de la boucle `for`. (Vous pouvez demander la valeur de `nbtours` interactivement via `cin` ou `scanf`, ou en CLI si vous souhaitez utiliser les paramètres `argc` et `argv` de `main`). Le code principal lance ensuite les threads. Chaque thread exécute le pseudo-algorithme suivant :

```
1  Exécute nbtours fois la séquence
2    Tâche()
3    Barrière()
```

Le pseudo-algorithme de $T\hat{a}che()$ est le suivant :

```
Affiche l'heure
Tire un nombre X aléatoirement dans U[10,20]
Calcule Fibonacci(X)
Calcule la durée passée pendant le calcul et l'affiche
```

Votre travail consiste surtout à designer/coder l'opération `Barrière()`. Vous êtes libre de procéder comme vous le souhaitez, à ceci près que vous n'avez pas le droit d'utiliser la structure `pthread_barrier_t`, ni la classe `std::barrier`. À vous de trouver comment fonctionne une barrière !

II.2 Travail à réaliser : récapitulatif et scripting

On aimerait pouvoir faire un tableau récapitulatif donnant pour chaque thread (colonne) l'heure de début, la valeur de X tirée, et la durée passée à faire le calcul (ligne). Il y aurait donc une entrée pour chaque itération. Le résultat serait présenté sous une forme similaire à celle donnée dans la table 1 (modulo le contenu des cases intérieures).

Thread	Iteration 1	Iteration 2	Iteration 3...
Thread **1**	(12 :00 :01, 12, 15)	(12 :00 :20, 45, 250)	(12 :10 :14, 1, 0.02)
Thread **2**	(12 :00 :01, 9, 7)	(12 :00 :19, 5, 4)	(12 :10 :20, 2, 0.04)

Table 1 – Exemple de sortie du script

— Proposer à votre encadrant la solution que vous pensez mettre en place.

— Écrire un script qui `grep` sur chaque thread et en tire les informations voulues afin d'afficher une belle synthèse : les threads passent-ils le même temps à calculer le même calcul ? Plus important, la barrière de synchronisation fonctionne t-elle comme prévu ?

1. Voir en bas de page de `https://graal.ens-lyon.fr/~ycaniou/Teaching/1415/L3/index.html`

III Lecteur/rédacteur (si temps)

— Rappelez les conditions de présence d'un problème de type Lecteur/rédacteur.
— Codez un moniteur capable de gérer un tel problème (attention à la propreté du code).