TP - ASR7 Programmation Concurrente

—

Contrôle continu final

Matthieu Moy, Fabien Rico, Adil Khalfa, Frédéric Suter

Printemps 2019

Afin d'obtenir tous les points il vous est demandé de justifier vos résultats. Le barème proposé est susceptible d'être modifié lors de la correction. Il n'est présent que pour vous donner une idée du poids relatif des différentes questions.

Pour toutes les questions, « Implémentez la fonction ... » signifie donner le prototype (types des arguments et de la valeur de retour) et le corps de la fonction. Les détails syntaxiques (point-virgules, arguments exacts des templates, `#include`...) ne sont pas pris en compte dans l'évaluation, mais essayez d'utiliser une syntaxe aussi proche que possible du C++.

Rappels sur C++11 et les threads

Pour vous aider, voici un rappel de la syntaxe C++11 pour les threads :

```
1  // Création et attente de terminaison d'un thread :
2  int main () {
3          // ...
4          std::thread t(f, 42, std::ref(x));
5          // ...
6          t.join();
7  }
8
9  // Déclaration d'un mutex
10 std::mutex m;
11
12 // Verrouillage/déverrouillage d'un mutex :
13 m.lock();
14 // ...
15 m.unlock();
16
17 // Instantiation d'un verrou :
18 {
19         std::unique_lock<std::mutex> l(m);
20         // ...
21 }
22
23 // Opérations sur une variable de condition :
24 std::condition_variable c;
```

```
25      c.wait(l); // l de type verrou (std::unique_lock par exemple)
26      c.notify_one();
27      c.notify_all();
28
29      // Opérations sur une variable de condition :
30      std::condition_variable_any c;
31      c.wait(m); // m de type mutex
32      c.notify_one();
33      c.notify_all();
```

I Programmation avec les threads (2 points)

On considère le programme suivant :

```
1       #include <thread>
2       #include <iostream>
3       #include <mutex>
4       using namespace std;
5       #define N 100
6
7       int v = 0;
8       mutex m1, m2;
9
10      void incr(int &v) {
11              for (int i = 0; i < N; ++i) {
12                      m1.lock();
13                      m2.lock();
14                      ++v;
15                      m2.unlock();
16                      m1.unlock();
17              }
18      }
19
20      void decr(int &v) {
21              for (int i = 0; i < N; ++i) {
22                      m2.lock();
23                      m1.lock();
24                      --v;
25                      m1.unlock();
26                      m2.unlock();
27              }
28      }
29
30      int main() {
31              thread t1(incr, ref(v));
32              thread t2(decr, ref(v));
33              t1.join();
34              t2.join();
35              cout << v << endl;
36      }
```

Q.I.1) - (1 point) Quel problème de synchronisation, vu en cours, pose ce programme ? Comment peut-il se manifester en pratique ? À quoi est-il du ?

Il y a un deadlock : on verrouille m1 et m2 dans des ordres différents dans les deux threads, on peut se retrouver dans la situation où un thread a verrouillé m1, l'autre m2, et les deux attendent mutuellement que l'autre déverrouille. En pratique le programme se bloque et l'affichage ne se produit jamais.

Q.I.2) - (1 point) Proposez une solution, et justifier en quoi elle résout le problème.

Deux solutions :

— N'utiliser qu'un seul mutex !

— Verrouiller dans le même ordre dans les deux threads.

II Ordonnancement sous Linux (4 points)

Dans cet exercice, nous allons utiliser l'implantation POSIX 1003b sur Linux. Il existe 100 niveaux de priorité :

— Le niveau 0 est réservé à `SCHED_OTHER` et les niveaux de priorité 1 à 99 aux politiques `SCHED_FIFO` et `SCHED_RR`.

— Les tâches de priorité 99 sont les tâches de plus forte priorité.

— `SCHED_OTHER` est dédié à l'ordonnanceur temps partagé.

— Le quantum utilisé par la politique `SCHED_RR` est de deux unités de temps.

— Quand une tâche `SCHED_FIFO` ou `SCHED_RR` arrive alors qu'il y a déjà d'autres tâches de même priorité en attente, la nouvelle tâche est insérée en fin de liste.

— L'ordonnancement est préemptif.

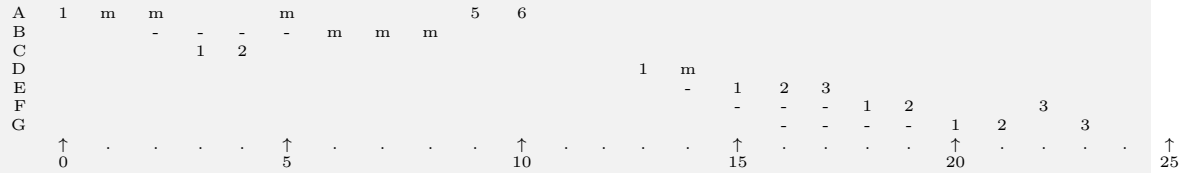
— A et D partagent un mutex, `m`.

Soit le jeu de tâches apériodiques suivant :

Tâche	Date d'arrivée	Priorité	Durée	Politique	Remarque
A	0	0	6	SCHED_OTHER	Verrouille le mutex m après 1 unité de temps, et le déverrouille 3 unités de temps de calcul plus tard
B	2	5	3	SCHED_FIFO	Verrouille le mutex m pendant l'ensemble de son exécution.
C	3	1	2	SCHED_FIFO	
D	13	8	2	SCHED_FIFO	Verrouille le mutex m après 1 unité de temps, et le déverrouille 1 unité de temps plus tard (donc juste avant de terminer)
E	14	2	3	SCHED_RR	
F	15	1	3	SCHED_RR	
G	16	1	3	SCHED_RR	

Q.II.1) - (3 points) Dessinez l'ordonnancement généré par l'ordonnanceur (vous pouvez utiliser la feuille de réponse en fin de sujet, n'oubliez pas votre numéro d'anonymat).

Q.II.2) - (1 point) Expliquez ce qu'est une inversion de priorité. Y en a t-il une sur cet exemple ?



On a une inversion de priorité aux instants 3 et 4 : B et C sont toutes les deux arrivées, ne partagent pas de mutex, B est plus prioritaire que C mais c'est C qui s'exécute.

III Administration système

III.1 Installation de logiciel (3 points)

Vous disposez de deux machines tournant sur la distribution Ubuntu (dérivée de Debian). Vous exécutez un script sur ces deux machines, il s'exécute correctement sur la machine A, mais sur la machine B il s'arrête sur l'erreur :

```
command not found: a2x
```

Malheureusement, il n'existe pas de paquet nommé `a2x` dans la distribution.

Q.III.1) - (1 point) Quelle commande peut-on lancer, sur la machine B, pour trouver un paquet en rapport avec le programme `a2x` ?

Malheureusement, la commande en question ne trouve rien. Mais vous ne désespérez pas : vous pouvez utiliser la machine A pour trouver l'information.

Q.III.2) - (1 point) Quelles commandes doit-on entrer pour trouver quel paquet contient l'exécutable `a2x` sur la machine A ? Vous aurez probablement besoin de deux commandes : pour trouver le chemin complet de l'exécutable, et pour trouver le paquet.

La commande en question vous apprend que `a2x` fait partie du paquet `asciidoc-base`.

Q.III.3) - (1 point) Quelle(s) commande(s) doit-on entrer pour installer le paquet `asciidoc-base` sur la machine B ?

III.2 Mises à jour (1 point)

Q.III.4) - (0,5 point) Votre voisin de bureau tente de faire les mises à jour sur son serveur qui tourne sous Ubuntu. Il lance la commande `apt upgrade` et est surpris de ne voir aucune mise à jour à faire. Qu'a-t-il oublié ?

`apt update`, qui télécharge la liste des paquets disponibles dans la dernière version.

Q.III.5) - (0,5 point) Une fois l'oubli ci-dessus réparé, que va faire précisément la commande `apt upgrade` (citez au moins deux étapes) ?

> Identifier les paquets qui ont besoin d'une mise à jour, télécharger les nouvelles versions de tous ces paquets, puis les installer (et accessoirement poser des questions à l'utilisateur).

III.3 Droits et utilisateurs (3 points)

Vous travaillez à plusieurs sur le même répertoire (tous sur la même machine, ou bien via un partage NFS). Pour l'instant, tous les fichiers ont les droits (en octal) 600 et tous les répertoires ont les droits 700, et vous êtes le propriétaire de l'ensemble. Votre collègue, qui ne fait pas partie du même groupe que vous, aimerait avoir un accès en lecture dans `mes_projets/tp_avec_Bob`, mais vous disposez aussi d'un répertoire `mes_projets/photos_de_soiree_avec_Charlie` dont vous souhaitez garder l'existence secrète.

Q.III.6) - (1 point) À quoi correspondent les droits 700 et 600 ?

Q.III.7) - (1 point) Quelle commande entrer pour donner les droits à votre collègue sur le répertoire `mes_projets/tp_avec_Bob` et son contenu ?

Q.III.8) - (1 point) Quelle commande entrer pour donner assez de droits à votre collègue sur `mes_projets/` pour qu'il puisse accéder à `mes_projets/tp_avec_Bob`, en gardant secret l'existence de `mes_projets/photos_de_soiree_avec_Charlie` ?

IV Programmation concurrente : les futurs (11 points)

Une construction intéressante pour écrire des programmes parallèles, mais non vue en cours de L3, est la notion de futur. Nous allons la découvrir ensemble et l'utiliser pour paralléliser un calcul de la fonction Fibonacci, définie par :

```
1  // Calcul naif, mais on s'interdira de l'optimiser
2  // autrement qu'en parallélisant.
3  uint64_t fibo(uint64_t i) {
4          if (i <= 1) {
5                  return 1;
6          }
7          return fibo(i - 1) + fibo(i - 2);
8  }
```

(Le type `uint64_t` est un entier non-signé sur 64 bits ; ce n'est pas très important ici on aurait pu écrire `int` à quelques détails près)

Si on veut calculer `fibo` sur plusieurs entrées différentes, on peut faire, en séquentiel :

```
1  int main() {
2          uint64_t v1 = fibo(12);
3          uint64_t v2 = fibo(42);
4          cout << v1 << ", " << v2 << endl;
5  }
```

Une alternative est de lancer les deux calculs, d'attendre qu'ils soient terminés, puis de récupérer les résultats.

Q.IV.1) - (2 points) Écrivez un programme `main` qui fait les mêmes calculs que la version séquentielle ci-dessus, mais en lançant les calculs `fibo(12)` et `fibo(42)` en parallèle, en faisant l'affichage après la fin des deux calculs. Vous aurez sans doute besoin de définir une fonction `fibo_ref(uint64_t i, uint64_t &result)`, et de créer deux `threads`. Pour l'instant la notion de future n'intervient pas.

Le principe du « futur » est le suivant : le lancement du calcul crée un thread, et produit un objet « futur », qui contiendra la valeur une fois qu'elle sera calculée. L'objet de type `Future` a une primitive `get` qui fait :

— Si la donnée n'a pas encore été produite (c-à-d. si le calcul associé n'est pas terminé), alors la fonction attend jusqu'à ce que la donnée soit disponible.

— Quand la donnée est disponible, la fonction `get` renvoie le résultat du calcul.

Dans tous les cas, quand la fonction `get` termine, elle renvoie la valeur résultat du calcul. On peut donc écrire le programme suivant, qui fait les mêmes calculs que ci-dessus, mais en exécutant les deux appels à `fibo` en parallèle :

```
1  int main() {
2          // Lance le calcul fibo(12) dans un thread, en crée un futur
3          // f1 qui contiendra la valeur quand le calcul sera terminé.
4          Future f1(fibo, 12);
5
6          // Idem pour fibo(42)
7          Future f2(fibo, 42);
8
9          // Les deux calculs tournent en parallèle avec ceci
10         cout << "Future created" << endl;
11
12         // Les appels à get() attendent que la valeur soit prête, et
13         // renvoient les résultats.
14         cout << f1.get() << ", " << f2.get() << endl;
15 }
```

On peut remarquer que le constructeur de `Future` prend en paramètre une fonction (comme le constructeur de `std::thread`). Pour vous aider, voici un petit programme manipulant des pointeurs de fonctions :

```
1  #include <iostream>
2  using namespace std;
3
4  typedef int (*function_t)(int);
5
6  void run_and_display(function_t f, int x) {
7          // Appel de la fonction f passée en paramètre :
8          int result = f(x);
9          cout << result << endl;
10 }
11
12 int increment(int arg) {
13         return arg + 1;
14 }
15
16 int main() {
```

```
17              // On passe la fonction 'increment' en paramètre de run_and_display :
18              run_and_display(increment, 42); // affiche 43
19      }
```

IV.1 La classe Future

Nous allons maintenant proposer une implémentation de la classe `Future`. Pour simplifier, on ne considère que des calculs qui prennent un paramètre `uint64_t` et renvoient un `uint64_t`, comme `fibo` ci-dessus [1].

La classe contient 2 méthodes et un constructeur :

- La méthode `uint64_t get()`, décrite ci-dessus.
- Une méthode void `resolve(uint64_t value)`, qu'il faudra appeler quand le calcul sera terminé. L'argument de resolve est la valeur résultat du calcul.
- Le constructeur, `Future(function_t f, uint64_t x)`, lance le calcul dans un thread. Il faut faire en sorte que la méthode `resolve(uint64_t value)` soit appelée à la fin de l'exécution de `f`. On pourra s'aider de la fonction suivante :

```
1       void run(Future &p, function_t f, uint64_t x) {
2               uint64_t result = f(x);
3               p.resolve(result);
4       }
```

Attention à ne pas utiliser `thread::join()` là où il ne faut pas (le calcul doit continuer après la fin de l'exécution du constructeur). On pourra par contre utiliser `thread::detach()` qui permet de dire explicitement que le thread peut continuer son exécution après la fin de la porté de la variable thread.

La classe `Future` est un moniteur de Hoare.

Q.IV.2) - (1 point) Donnez les champs de la classe `Future`

Q.IV.3) - (1,5 point) Donnez le constructeur de la classe

Q.IV.4) - (1 point) Donnez le corps de la méthode `get`

Q.IV.5) - (1 point) Donnez le corps de la méthode `resolve`

IV.2 Utilisation pour accélérer `fibo`

On souhaite maintenant accélérer le calcul de `fibo` a proprement parler, en utilisant notre classe `Future` (sans changer l'algorithme de calcul : on gardera les deux appels récursifs).

Le calcul séquentiel de la fonction ressemble à ceci :

```
1       // ...
2       uint64_t a = fibo(i - 1); // (A)
3       uint64_t b = fibo(i - 2); // (B)
4       return a + b;
```

La version parallèle s'appelera `fibo_par`, et lancera les calculs (A) et (B) en parallèle. Pour éviter de créer trop de threads, le calcul `fibo_par(n)` pour `n <= 40` sera fait séquentiellement (autrement dit, utilisera le même algorithme que `fibo(n)`), et pour `n > 40`, chaque appel à `fibo_par(n)` créera deux `Future` dans lesquels les appels récursifs (A) et (B) seront exécutés (en utilisant la fonction `fibo_par`, qui elle-même créera potentiellement plus de parallélisme).

1. Mais libre à vous si vous voulez donner une solution plus générale à base de templates variadiques, ...

Q.IV.6) - (1,5 point) Donnez le corps de la fonction `uint64_t fibo_par(uint64_t i)`

Q.IV.7) - (1 point) Proposez une version plus efficace, sur le même principe mais en n'utilisant qu'un `Future` au lieu de deux. Pourquoi est-ce plus efficace ?

IV.3 Pour aller plus loin

Notre implémentation des `Future`s crée un thread par calcul. C'est assez inefficace, et on préfèrerait instancier un nombre fixe de « worker threads » (typiquement un thread par cœur de la machine hôte), à qui on enverrait le travail à effectuer.

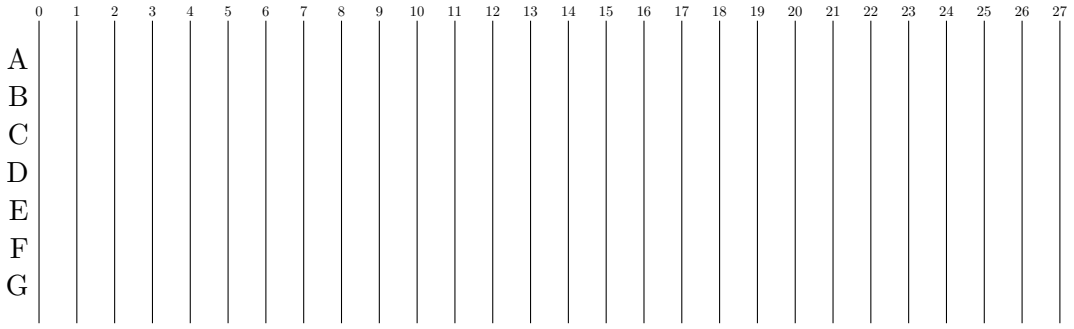
Q.IV.8) - (2 points) Quelle construction vue en cours pourrait-on utiliser pour cela ? Expliquez brièvement, de préférence en vous appuyant sur un ou plusieurs schémas, comment le système fonctionnerait.

V Feuille de réponse

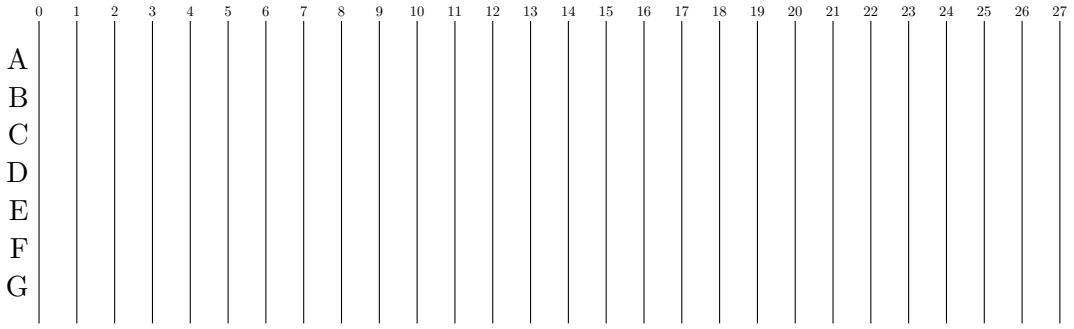
Numéro d'anonymat : .....

Un seul des tableaux doit être rempli, les autres peuvent servir de brouillons ou de secours en cas d'erreur. **Un seul sera corrigé**, barrez distinctement les brouillons.

Brouillon :



Brouillon :



Question II.1 :