

Synchronisation

Exercice 1. Questions de cours

Les questions de cours sont à destination de vous permettre de vérifier votre compréhension du cours. Elles sont à travailler à l'avance et ne seront pas traitées en TD ou TP.

1. Qu'est-ce qu'une section critique ?
2. Quelles sont les propriétés attendues d'une section critique ?
3. Quels sont les inconvénients du Mutex ?

Exercice 2. Création de threads

La création de threads se fait à l'aide de l'appel système `pthread_create`. Contrairement à l'appel système `fork`, il prend en argument une fonction qui sera exécutée dans un thread séparé et un éventuel argument à donner à cette fonction. Il est possible de configurer certains paramètres du thread créé via un autre argument (*attr*) que nous n'utiliserons pas ici et que l'on va donc toujours laisser à `NULL`.

Avant de commencer, lisez la documentation des appels systèmes suivant : `pthread_create` et `pthread_join`.

Afin d'utiliser les appels systèmes de manipulation des threads, il est nécessaire d'inclure le fichier d'en-tête `pthread.h`. Sur certains systèmes, au moment de la compilation il peut aussi être nécessaire d'ajouter l'option `-l pthreads`.

1. Écrivez en C un programme qui crée 10 threads et attend la fin de leur exécution. Chacun de ces threads affichera "Bonjour" avant de se terminer.
2. Modifiez votre programme de manière à passer à chacun des threads un numéro qui l'identifie. Chaque thread devra afficher le numéro qu'il a reçu.

Exercice 3. Recherche parallèle

Dans l'exercice 4 du td 2, nous avons implémenté la recherche en parallèle d'une valeur dans un tableau à l'aide de processus. Pour une tâche aussi simple, l'utilisation de processus est en général très inefficace car les processus sont lourds à créer, les threads sont plus adaptés.

1. Reprenez le code que vous avez écrit pour la première question de l'exercice 4 du td 2 et modifiez-le pour que le tableau soit alloué dans une variable globale et que la recherche soit faite dans une fonction séparée.
2. Paralléliser ce code en utilisant 2 threads. Vous devez passer à chaque thread un identifiant qui lui permette de calculer la partie du tableau dans laquelle il doit effectuer la recherche.

Exercice 4. Synchronisation simple

Comme on l'a vu en cours, si les threads ont l'avantage de rendre le partage de données très simple et efficace, il est nécessaire de faire extrêmement attention car il n'offrent aucune protections automatique lors d'accès concurrent à une même donnée.

1. Écrivez un programme avec une variable entière globale qui lance 10 threads et attend la fin de leur exécution avant d'afficher la valeur de cette variable. Chaque thread exécutera simplement une boucle qui incrémente dix mille fois la variable. Lancer votre programme plusieurs fois afin d'observer le résultat.

Attention, le code que nous écrivons ici est très naïf et le compilateur risque de réaliser des optimisations qui vont artificiellement cacher l'effet que l'on souhaite mettre en évidence. Afin d'éviter ce problème, il est possible de désactiver les optimisations sur la variable globale en la déclarant volatile :

```
volatile int count = 0;
```

2. Le problème de synchronisation se trouve au niveau de l'incrément du compteur, il est nécessaire de la protéger afin qu'un seul thread à la fois puisse accéder à cette variable. Modifiez votre code manière à utiliser une exclusion mutuelle pour protéger ces accès lorsque cela est nécessaire.

Les appels systèmes principaux nécessaires pour l'utilisation des exclusions nmuetuelles en C sont: `pthread_mutex_lock`, `pthread_mutex_unlock` et `pthread_mutex_init`, ainsi que la constante `PTHREAD_MUTEX_INITIALIZER`.

Exercice 5. Lecteurs-écrivains

Le problème des lecteurs-écrivains est un problème classique de synchronisation. On considère une base de données et un ensemble de threads qui souhaitent lire (les *lecteurs*) et écrire (les *écrivains*) dans cette base. Chaque thread attend un temps aléatoire puis demande à acquérir la base pour lire ou pour écrire. Le travail sur la base dure un certain temps puis le thread libère la base. On souhaite réunir les conditions suivantes :

- C1. Plusieurs lecteurs peuvent accéder à la base en lecture en même temps ;
- C2. Un seul écrivain peut accéder à la base à la fois ;
- C3. Un écrivain ne peut pas accéder à la base en même temps que des lecteurs.

Chaque lecture ou écriture dans la base prend un temps variable, pour la suite de cet exercice, on les simulera à l'aide des deux fonctions suivantes :

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 void lire() {
5     printf("lire debut\n");
6     usleep(rand() % 500000);
7     printf("lire fin\n");
8 }
9 void ecrire() {
10    printf("ecrire debut\n");
11    usleep(rand() % 500000);
12    printf("ecrire fin\n");
13 }
```


Vous testerez votre code en lançant à chaque fois plusieurs threads en parallèle. Vous êtes encouragés à expérimenter avec différentes quantités de threads et comparer les résultats.

1. La fonction suivante va aléatoirement réaliser des lectures et des écritures dans la base. Si elle est exécutée par plusieurs threads en même temps, les conditions C1, C2 et C3 sont-elles satisfaites ?

```

1 void *run(void *arg) {
2     while (1) {
3         if ((rand() % 2) == 0)
4             lire();
5         else
6             ecrire();
7     }
8 }
```

2. Proposez une première solution utilisant une section critique satisfaisant les conditions C2 et C3. (dans cette première version un seul lecteur est autorisé à accéder à la base en même temps)

 Les questions suivantes de cet exercice sont difficiles. Trouver une solution qui marche correctement dans tous les cas est difficile et il est encore plus difficile de le prouver. Vous êtes encouragés à travailler ces questions afin de proposer des solutions puis d'étudier leurs corrections afin de comprendre comment les résoudre correctement. La synchronisation de threads est un sujet difficile que l'on ne peut maîtriser que par la pratique et l'expérience.

3. Proposez maintenant une solution au problème respectant les trois conditions. (Un seul verrou ne sera pas suffisant ici)
4. Votre solution prévient-elle du risque de famine ? Proposez si besoin une solution à ce problème. (on parle de famine si un thread peu se retrouver à attendre un temps infini alors que les autres threads progressent.)