TD11. Concurrence

Exercice 1. Entrelacements

On suppose qu'on dispose de plusieurs programmes compteurs qui affichent le compte dans leur langue :

- un compteur numérique qui compte 1,2,3,...
- un compteur latin qui compte a,b,c,d...
- un compteur grec qui compte α, β, γ, ...
- un aviateur qui compte alpha, bravo, charlie, delta, echo ...

On peut lancer ces compteurs en parallèle, en leur demandant de compter jusqu'à une certaine valeur. Chacun suit son compte, à son rythme, mais on n'a aucune hypothèse sur leurs rythmes relatifs.

1. Quels affichages est-on susceptible d'observer si on lance latin(2) et grec(1) ?

2. Quel affichage est-on susceptible d'observer si on lance num(2), latin(2) et aviateur(1) ?

3. Donner quelques affichages possibles si on lance latin(3) et grec(3), mais qu'on leur demande de s'attendre après avoir affiché 2 dans leur langue. Donner le nombre d'affichages possibles.

Exercice 2. Identifier une section critique

1. En exécutant le programme suivant, quels sont tous les affichages possibles ?

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

int compteur = 0;

void* incremente(void* arg) {
    for (int i = 1; i < 3; i++){
        compteur = compteur + i;
    }
    return NULL;
}

int main(){
    pthread_t tA, tB, tC;

    pthread_create(&tA, NULL, incremente, NULL);
    pthread_create(&tB, NULL, incremente, NULL);

    pthread_join(tA, NULL);
    pthread_join(tB, NULL);

    printf("%d", compteur);
}
```

2. Modifier le programme pour que le seul affichage possible soit 6.

Exercice 3. Le problème du rendez-vous
Dans cet exercice, on se penche sur des problèmes de rendez-vous entre plusieurs fils d'exécution. La solution proposée diffère selon le nombre de fils d'exécution qui doivent se coordonner et le nombre de fois qu'ils vont se donner rendez-vous.

A. Deux fils d'exécution se rencontrent une fois

On suppose qu'on souhaite synchroniser deux threads P_1 et P_2 entre l'exécution de leurs premières instructions, rassemblées sous la notation A, et le reste de leurs instructions, rassemblées sous la notation B. Autrement dit, on a la situation suivante :

Algo. du fil d'exécution P_1
1 A_1 ;
2 RDV avec P_2 ;
3 B_1 ;

Algo. du fil d'exécution P_2
1 A_2 ;
2 RDV avec P_1 ;
3 B_2 ;

1. Proposer une solution qui permette de synchroniser deux fils d'exécution une fois à l'aide de sémaphores.

B. Plusieurs fils d'exécution se rencontrent une fois

Dans cette section on suppose qu'on a N fils d'exécution $(P_i)_{1\leq i\leq N}$ à synchroniser. Comme précédemment on note A_i (resp. B_i) les instructions que P_i doit réaliser avant (resp. après) le rendez-vous.
Pour réaliser cette synchronisation, on crée un objet qu'on appellera une barrière. Cette barrière `b` est connue des N fils d'exécution, qui peuvent alors la solliciter pour passer à travers la fonction `appel_barriere (b)`, comme les piétons appellent le feu vert avant de traverser une route. Lorsqu'un fil d'exécution P_i appelle cette fonction, c'est qu'il est au rendez-vous : il a fini A_i et il attend les autres fils d'exécution pour passer la barrière et faire B_i après.
2. Proposer une implémentation en pseudo-code d'un tel objet barrière et de la fonction `appel_barriere`. On pourra utiliser verrous et sémaphores. On veillera aussi à expliciter comment un tel objet doit être initialisé dans une fonction `cree_barriere`.

C. Plusieurs fils d'exécution se rencontrent plusieurs fois

Dans la section précédente, on a proposé une solution pour que $N \in \mathbb{N}^*$ se rencontrent une fois. Cependant on peut aisément imaginer des algorithmes où une famille de fils d'exécution doit se réunir à chaque tour de boucle.
3. Que dire de la solution qui consiste à créer, avant de lancer les fils d'exécution, une barrière `b` comme proposé ci-avant, et de demander à chaque fil d'exécution d'attendre les autres à chaque tour en appelant à `attend_barriere(b)` ? Est-elle robuste ? Plus précisément, cette solution convient-elle si les fils d'exécution ne s'exécutent pas tous autant de fois ?
4. Proposer une amélioration de l'implémentation de barrière proposée ci-avant qui soit robuste à la disparition de certains fils d'exécution initialement rattachés à la barrière. On pourra supposer qu'il s'agit plus de départ volontaire que de disparition, et que les fils d'exécution se signalent avant de disparaître.

Exercice 4. Le barbier endormi

La boutique du barbier est composée d'une salle d'attente contenant N chaises et du salon où se trouve la chaise du barbier. Lorsque le barbier a fini de raser un client, il fait entrer le client suivant dans le salon. Si la salle d'attente est vide, le barbier s'y installe pour dormir. Si un client trouve le barbier endormi, il le réveille. Si non, il s'installe dans la salle d'attente s'il reste de la place (et rentre chez lui sinon).

1. Écrivez le code du client et du barbier sans vous préoccuper de synchronisation, mais simplement des opérations que veulent réaliser les processus. En particulier, ignorez pour l'instant que le barbier dort parfois.

2. Il s'agit maintenant d'ajouter les synchronisations nécessaires au programme écrit à la question précédente. Le premier problème à résoudre est une condition de compétition entre les clients lorsqu'ils rentrent dans la salle d'attente. Corrigez ce problème.

3. Assurez-vous ensuite que le barbier ne commence pas à couper les cheveux tant que le client n'est pas prêt.

4. Enfin, assurez-vous ensuite que le client ne s'assoit pas sur le siège tant que le barbier n'est pas prêt.