

TP 2 – Exclusion Mutuelle par les Mutex

L'objectif de ce TP est de réaliser l'exclusion mutuelle entre threads concurrents en utilisant les Mutex de la norme Posix sous Linux, qui sont destinés à l'exclusion mutuelle. Un mutex est un sémaphore particulier destiné uniquement à l'exclusion mutuelle (MUTual Exclusion). Il peut être assimilé à un verrou pour contrôler l'accès à une ressource partagée.

Un Mutex peut être dans deux états : déverrouillé (pris par aucun thread) ou verrouillé (appartenant à un thread). Un Mutex ne peut être pris que par un seul thread à la fois. Un thread qui tente de verrouiller un Mutex déjà verrouillé est bloqué jusqu'à ce que le Mutex soit déverrouillé. On peut assimiler un Mutex à un sémaphore binaire.

On manipule les Mutex grâce aux fonctions suivantes :

#include <pthread.h>

pthread_mutex_t Mutex = PTHREAD_MUTEX_INITIALIZER ;

int pthread_mutex_lock(pthread_mutex_t *Mutex);

int pthread_mutex_unlock(pthread_mutex_t *Mutex);

PTHREAD_MUTEX_INITIALIZER est une constante utilisée pour initialiser le Mutex pointé par la variable Mutex.

pthread_mutex_lock verrouille le Mutex. Si le Mutex est déverrouillé, il devient verrouillé. Si le Mutex est déjà verrouillé par un autre thread, cette fonction bloque le thread appelant. Cette fonction a le même effet que la primitive P sur un sémaphore binaire.

pthread_mutex_unlock déverrouille le Mutex (le rend à l'état libre) et réveille le thread bloqué s'il y en a. Cette fonction a le même effet que la primitive V sur un sémaphore binaire.

Exercice 1 : Il s'agit ici d'utiliser un Mutex pour protéger l'accès à une variable partagée.

- Reprendre le programme de l'exo1 du TP1.
- Résoudre le problème de l'accès à la variable partagée en utilisant un Mutex.

- Pour compiler cet exercice il faut utiliser la commande : `gcc exo1.c -lpthread`
- Pour lancer l'exécution du programme il faut taper la commande `./a.out`

Exercice 2 : Ecrire deux versions d'un programme qui crée 10 threads numérotés de 1 à 10 et travaillant tous sur une même variable partagée Nb. Chaque thread affiche d'abord la valeur de Nb, puis la multiplie par son numéro de thread, puis affiche le résultat.

1. Dans la première version du programme, l'accès d'un thread à Nb doit être exclusif depuis sa lecture jusqu'à son affichage.
2. Dans la deuxième version du programme, seul l'accès à Nb est réalisé en exclusion mutuelle, les autres tâches des threads sont réalisées en totale concurrence entre eux (Utilisez la primitive sleep entre le premier affichage de Nb et le calcul.)

Quelles différence y a-t-il entre ces deux solutions ?