

TP 1 – Exclusion Mutuelle par attente active

Exercice 1 : Problème des accès concurrents

Ecrire un programme qui initialise une variable entière V à 5000000 et crée deux thread th1 et th2. Le thread th1 incrémente V 5000000 fois et le thread th2 décrémente V 5000000 fois.

Quelle devrait être la valeur finale de V ? Que remarquez-vous après plusieurs exécutions du programme ? Expliquez.

Exercice 2 : algorithme de Dekker

Réécrire le programme précédent en y intégrant la solution de Dekker pour résoudre le problème de l'accès concurrent à la même variable.

A rendre la séance prochaine :

Réécrire le programme précédent en y intégrant la solution de Peterson pour résoudre le problème de l'accès concurrent à la même variable.

Rappels des fonctions de manipulation des threads

```
#include <pthread.h>
```

```
int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void  
                    *(*routine)(void*), void *arg);
```

```
int pthread_exit (void *value_ptr) ;
```

```
int pthread_join (pthread_t *thread, void **value_ptr) ;
```

- pthread_create : crée un thread, et renvoie 0 si la création s'est bien déroulée, ou le code de l'erreur sinon Elle reçoit en argument :

- ⤴ le TID (Thread Identifier) du thread,
- ⤴ une constante
- ⤴ un pointeur vers la fonction exécutée par le thread
- ⤴ un argument de la fonction

- pthread_exit : permet de quitter un thread, elle ne renvoie aucune valeur, et attend un seul argument qui est l'adresse de la variable à renvoyer au programme qui a appelé le thread

- pthread_join : bloque l'appelant en attente de la fin du thread passé en 1^{er} argument, alors que le 2nd argument est un pointeur de pointeur qui servira à récupérer l'adresse de la valeur renvoyée par le thread_exit du thread, cette fonction renvoie 0 si le thread se termine correctement, sinon elle renvoie le code de l'erreur