
TP 02 – Parallélisme et synchronisation

Exercice 1.*valeur retournée par un thread*On se basera sur le fichier `thread_creation.c`.

1. Écrire le programme `thread_creation_retour.c` dont l'exécution :

- crée un thread en lui passant le paramètre entier 21 ;
- récupère la valeur de retour du thread (un entier) et l'affiche.

Le thread exécute une fonction qui prend un paramètre entier, et retourne cet entier plus 21.

Aide : seul le thread a besoin de faire un appel à `malloc`.**Exercice 2.***6 threads inutiles... ou pas !*On se basera sur `thread_creation.c` pour créer le programme `thread_glob.c`.

1. Ajouter une variable globale `glob` initialisée à 0.

2. Le programme doit maintenant :

- Afficher la valeur de `glob` ;
- Lancer 6 threads (sans argument) :
 - 3 exécutent la fonction `increment` ;
 - 3 exécutent la fonction `decrement` ;
- Attendre les 6 threads dans leur ordre de lancement ;
- Afficher le message :

« ce superbe programme ne sert a rien car `glob` vaut toujours `<glob>` »La fonction `increment` affiche « THREAD : increment » puis fait une boucle de 10 000 fois l'instruction `glob = glob + 1;`.La fonction `decrement` affiche « THREAD : decrement » puis fait une boucle de 10 000 fois l'instruction `glob = glob - 1;`.**Exercice 3.***mutex*

1. Implémenter un mécanisme d'exclusion mutuelle pour protéger la variable `glob` dans l'Exercice 2.

Exercice 4.*sémaphore*

Un thread :

- prend un sémaphore avec l'appel `prendre_semaphore(&mon_mutex_semaphore)`,
- rend un sémaphore avec l'appel `rendre_semaphore(&mon_mutex_semaphore)`.

1. Implémenter un sémaphore pour qu'au plus cinq threads puissent entrer en section critique.

Un thread lecteur exécute la fonction `lecteur` et

- prend un sémaphore avec `prendre_semaphore_lecteur(&mon_mutex_semaphore)`,
- rend un sémaphore avec `rendre_semaphore_lecteur(&mon_mutex_semaphore)`.

Un thread rédacteur exécute la fonction `redacteur` et

- prend un sémaphore avec `prendre_semaphore_redacteur(&mon_mutex_semaphore)`,
- rend un sémaphore avec `rendre_semaphore_redacteur(&mon_mutex_semaphore)`.

2. Implémenter un sémaphore pour qu'un thread rédacteur ne puisse entrer en section critique que si aucun thread lecteur n'est en section critique.

Exercice 5.*moniteur : mutex en LIFO*

Nous allons écrire un moniteur qui implémente une file d'attente LIFO pour un mutex.
On utilisera le type suivant :

```
typedef struct mutex_lifo_t {  
    pthread_cond_t cond;  
    pthread_mutex_t mutex;  
    int lifo_tete;  
    int lifo_queue;  
} mutex_lifo_t;
```

Accessible grace aux méthodes :

```
void mutex_lifo_init(mutex_lifo_t* mutex_lifo);  
void mutex_lifo_lock(mutex_lifo_t* mutex_lifo);  
void mutex_lifo_unlock(mutex_lifo_t* mutex_lifo);
```

1. Écrire le code de ces trois méthodes.
2. Quel est l'inconvénient / l'avantage d'un mutex en LIFO ?
3. Écrire un programme complet pour tester votre mutex en LIFO.