TD : Les threads

Objectifs :

Mise en œuvre d'une application à base de threads.

On souhaite mettre en œuvre une application linux multi threads permettant de réaliser un traitement long, puis plusieurs traitements.
L'application doit permettre de remplir un tableau de 4096 doubles. Ces nombres représentent un signal. L'application devra activer ensuite un thread qui calcule la valeur minimale, la valeur maximale, la moyenne et la variance du signal. Une fois le calcul fini, ces valeurs seront placées dans cet ordre à la fin du tableau.
Pendant ce calcul le thread principal de l'application se bloquera en attente de la fin du thread afin d'afficher, après calculs, les résultats obtenus.

Partie 1 : Synchronisation par attente de fin du thread

1. Créez un programme "***Thread1.c***"
2. Écrire une fonction "***ThreadCalculs***", point d'entrée du thread.
3. Déclarez un tableau de 4100 doubles (4096 pour le signal et 4 pour les calculs décrits ci-dessus).
4. Codez la fonction main afin qu'elle réalise :
   - L'initialisation des 4096 premiers éléments du tableau de doubles avec les points représentatifs du signal $x(t) = Vo + Vc \sin(2 \pi f t)$ sur une période, soit pour i=0 à i=4096 : $x(i) = Vo + Vc \sin(2 \pi i / 4096)$;
   - La création du thread avec des attributs par défaut, en lui passant comme paramètre le tableau représentatif du signal ;
   - Le blocage sur la fin du thread ;
   - Lorsque le thread est terminé, l'affichage des différentes valeurs calculées par le thread ;
5. Codez la fonction "***ThreadCalculs***" afin de réaliser :
   - L'identification de la valeur minimale et de la valeur maximale du tableau passé en paramètre, puis le stockage dans l'ordre à la fin du tableau ;
   - Le calcul de la moyenne et de la variance des 4096 premiers éléments du tableau, puis le stockage de ces valeurs dans l'ordre à la fin du tableau, à la suite de la valeur maximale ;

→ Tester cette partie avec Vo = -2.3 et Vc = 1.7.

La variance d'une série de données indique à quel point les données sont distantes. Plus la variance est proche de zéro, plus les données sont proches l'une de l'autre.
On rappelle que la variance d'un tableau est :

$$Variance = \frac{1}{Taille} * \sum_{i=0}^{i=TAILLE-1} (Tab[i] - moyenne)^2$$

Partie 2 : Synchronisation par mutex

Le but de cette partie est de créer le thread avant l'initialisation du tableau. Il va falloir donc réaliser la synchronisation du départ du thread par un ***mutex_thread***. L'affichage des valeurs minimale, maximale, moyenne et variance se fera toujours après la fin du thread.

1. Créez un autre programme "***Thread2.c***" et copiez dedans le code du programme "***Thread1.c***" ;
2. Modifiez le "*main*" précédant de la manière suivante :
   - Déclarez une structure permettant de contenir toutes les variables à passer au thread de calculs ;
   - Ajoutez à la fonction *main* l'initialisation d'un "***mutex_thread***" verrouillé ;
   - Initialisez une variable du type de la structure définie auparavant afin de passer au thread tout ce qui lui est nécessaire ;
   - Créez le thread puis initialiser le tableau du signal ;
   - Complétez le code afin de gérer le "***mutex_thread***" ;
   - Ajoutez la suppression "***mutex_thread***" ;
   - Modifiez le code de la fonction "***ThreadCalculs***" pour tenir compte de la synchronisation par "***mutex_thread***" ;

→Tester cette partie

Partie 3 : Synchronisation par sémaphore POSIX et détachement du thread

L'utilisation des sémaphores POSIX de la bibliothèque ***LinuxThreads*** permet la synchronisation entre threads.
Le but de cette partie est de réaliser la même synchronisation que précédemment mais avec les sémaphores POSIX. Dans ce cas il est possible de choisir les attributs d'un thread. Il faudra détacher le thread de calcul et se synchroniser en utilisant les sémaphores POSIX. Il est évident qu'on ne pourra plus attendre la fin du thread de la même manière pour afficher les résultats, cependant on réalisera toujours la création du thread avant l'initialisation des données.

1. Créez un autre programme "***Thread3.c***" ;
2. Ajoutez l'initialisation du (ou des) sémaphores POSIX à la fonction *main* ;
3. Initialisez une variable du type de structure défini auparavant afin de passer au thread tout ce qui lui est nécessaire ;
4. Modifiez la fonction *main* afin de créer le thread, son détachement et enfin d'initialiser les données ;
5. Modifiez le code de la fonction main pour gérer ce (ou ces) sémaphore(s) POSIX ;
6. Ajoutez la suppression du (ou des) sémaphore(s) POSIX dans la fonction *main* ;
7. Modifier le code de la fonction "***ThreadCalculs***" pour tenir compte de la synchronisation par sémaphore POSIX ;

→ Tester cette partie

Partie 4 : Synchronisation de plusieurs threads

Dans cette partie, la détermination de la valeur minimale, de la valeur maximale et le calcul de la moyenne et de la variance peuvent se faire en parallèle.
Il est donc envisageable de séparer les deux recherches et le calcul de la moyenne et de la variance.
Vous allez donc créer 3 threads tous "détachés" :

- L'un pour la recherche de la valeur minimale,
- L'autre pour la recherche de la valeur maximale,
- Le dernier pour le calcul de la moyenne et de la variance,

Il faudra synchroniser tout cet ensemble à l'aide des sémaphores POSIX (2 sémaphores doivent suffire).

1. Créez un autre programme "***Thread4.c***" ;
2. Modifiez le code de la fonction main en conséquence ;
3. Codez la fonction "***threadValMinimale***" ;
4. Codez la fonction "***threadValMaximale***" ;
5. Codez la fonction "***threadMoyVar***" ;

→ *Tester cette partie*