Application

Imprimer

Précédent
Synchronisation des threads avec un mutex

Suivant
testTh.c

Imaginons un simple tableau d'entiers, rempli par un thread (lent) et lu par un autre (plus rapide). Le thread de lecture doit attendre la fin du remplissage du tableau avant d'afficher son contenu. Pour cela, nous pouvons utiliser des **mutex** afin de protéger le tableau pendant le temps de son remplissage.

```
nessim@nessim-virtual-machine:~$ gcc application.c -o application
nessim@nessim-virtual-machine:~$ ./application
Producteur : Ajout de 1 au tableau
Consommateur : En attente du remplissage du tableau...
Producteur : Ajout de 2 au tableau
Producteur : Ajout de 3 au tableau
Producteur : Ajout de 4 au tableau
Producteur : Ajout de 5 au tableau
Producteur : Ajout de 6 au tableau
Producteur : Ajout de 7 au tableau
Producteur : Ajout de 8 au tableau
Producteur : Ajout de 9 au tableau
Producteur : Ajout de 10 au tableau
Consommateur : Contenu du tableau :
1 2 3 4 5 6 7 8 9 10
nessim@nessim-virtual-machine:~$
```

Open — application.c — Save

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

#define TABLE_SIZE 10

// Variables partagées
int tableau[TABLE_SIZE];
int remplissage_termine = 0; // Indicateur de fin de remplissage
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // Mutex pour protéger les variables partagées
pthread_cond_t cond_remplissage = PTHREAD_COND_INITIALIZER; // Variable de condition

// Fonction exécutée par le thread producteur
void* remplir_tableau(void* arg) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        pthread_mutex_lock(&mutex); // Verrouiller le mutex
        tableau[i] = i + 1; // Remplir le tableau avec des valeurs
        printf("Producteur : Ajout de %d au tableau\n", tableau[i]);
        pthread_mutex_unlock(&mutex); // Déverrouiller le mutex
        sleep(1); // Simuler un remplissage lent
    }

    pthread_mutex_lock(&mutex);
    remplissage_termine = 1; // Marquer le remplissage comme terminé
    pthread_cond_broadcast(&cond_remplissage); // Signaler tous les threads en attente
    pthread_mutex_unlock(&mutex);

    return NULL;
}

// Fonction exécutée par le thread consommateur
void* lire_tableau(void* arg) {
    pthread_mutex_lock(&mutex);
    while (!remplissage_termine) {
        printf("Consommateur : En attente du remplissage du tableau...\n");
        pthread_cond_wait(&cond_remplissage, &mutex); // Attendre la fin du remplissage
    }
    pthread_mutex_unlock(&mutex);


    // Afficher le contenu du tableau
    printf("Consommateur : Contenu du tableau :\n");
    for (int i = 0; i < TABLE_SIZE; i++) {
        printf("%d ", tableau[i]);
    }
    printf("\n");

    return NULL;
}

int main() {
    pthread_t thread_producteur, thread_consommateur;

    // Créer les threads
    pthread_create(&thread_producteur, NULL, remplir_tableau, NULL);
    pthread_create(&thread_consommateur, NULL, lire_tableau, NULL);

    // Attendre la fin des threads
    pthread_join(thread_producteur, NULL);
    pthread_join(thread_consommateur, NULL);

    // Nettoyer les ressources
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond_remplissage);

    return 0;
}
```

C — Tab Width: 8 — Ln 66, Col 14 — INS