Philosophers pthread fonctions

Voici une explication détaillée de chacune des fonctions que tu mentionnes, leur rôle, les arguments qu'elles prennent et les valeurs qu'elles renvoient.

---

1. `pthread_mutex_init`

Cette fonction est utilisée pour initialiser un mutex (un verrou). Elle permet de préparer un mutex à être utilisé dans un programme multithread pour synchroniser l'accès aux ressources partagées.

Arguments :

- `pthread_mutex_t *mutex` : Un pointeur vers le mutex que tu souhaites initialiser.
- `const pthread_mutexattr_t *attr` : Un pointeur vers des attributs de mutex (tu peux passer `NULL` pour utiliser les attributs par défaut).

Retour :

- `0` si l'initialisation réussit.
- `Erreur` si l'initialisation échoue (par exemple, en cas de ressources insuffisantes).

Exemple :

```c
pthread_mutex_t fork;
pthread_mutex_init(&fork, NULL);  // Initialise un mutex pour une fourchette
```

---

2. `pthread_create`

Cette fonction crée un nouveau thread (unité d'exécution parallèle). Elle associe un thread à une fonction que ce thread va exécuter.

Arguments :

- `pthread_t *thread` : Un pointeur vers un identifiant de thread. C'est là que le système stockera l'identifiant du thread créé.
- `const pthread_attr_t *attr` : Un pointeur vers des attributs de thread (tu peux passer `NULL` pour les attributs par défaut).
- `void *(*start_routine)(void *)` : Le pointeur vers la fonction que le thread doit exécuter. Elle doit prendre un argument de type `void *` et retourner un `void *`.
- `void *arg` : Un argument que tu veux passer à la fonction d'exécution du thread (ici, un pointeur vers un philosophe).

Retour :

- `0` si le thread a été créé avec succès.
- `Erreur` si la création du thread échoue (par exemple, en raison de ressources insuffisantes).

Exemple :

```c
pthread_t thread;
pthread_create(&thread, NULL, routine, &philosophe);
```

Ici, la fonction `routine` sera exécutée par le thread, et l'argument `&philosophe` (un pointeur vers un philosophe) sera passé à cette fonction.

---

3. `pthread_join`

Cette fonction permet au thread principal (ou à un autre thread) d'attendre qu'un thread spécifique ait terminé son exécution. C'est utile pour synchroniser l'exécution des threads.

Arguments :

- `pthread_t thread` : L'identifiant du thread que tu veux attendre.
- `void **retval` : Un pointeur vers une variable où la fonction de thread peut stocker sa valeur de retour. Si tu n'as pas besoin de récupérer cette valeur, tu peux passer `NULL`.

Retour :

- `0` si l'attente a réussi.
- `Erreur` si l'attente échoue (par exemple, si le thread n'existe pas).

Exemple :

```c
pthread_join(thread, NULL);  // Attend que le thread `thread` termine
```

Cela garantit que le thread principal (ou un autre thread) attend que `thread` termine avant de continuer l'exécution.

---

4. `pthread_mutex_destroy`

Cette fonction détruit un mutex précédemment initialisé avec `pthread_mutex_init`. Elle est généralement utilisée pour libérer les ressources lorsque le mutex n'est plus nécessaire.

Arguments :

- `pthread_mutex_t *mutex` : Le pointeur vers le mutex à détruire.

Retour :

- `0` si la destruction réussit.
- `Erreur` si la destruction échoue (par exemple, si le mutex est encore verrouillé).

Exemple :

```c
pthread_mutex_destroy(&fork);  // Détruit un mutex (libère les ressources)
```

Résumé des rôles

1. `pthread_mutex_init` : Initialise un mutex pour protéger une ressource partagée.
2. `pthread_create` : Crée un thread et lui attribue une fonction (`routine`) à exécuter, avec un argument spécifique.
3. `pthread_join` : Attend qu'un thread termine son exécution avant de continuer l'exécution du programme principal ou d'un autre thread.
4. `pthread_mutex_destroy` : Détruit un mutex pour libérer les ressources associées à celui-ci.

Exemple complet pour tes threads de philosophes

Voici comment tu pourrais utiliser ces fonctions dans ton code pour gérer les philosophes et les mutex associés aux fourchettes :

```c
pthread_mutex_t forks[5];   // Mutex pour les 5 fourchettes
pthread_t threads[5];       // Threads pour les 5 philosophes

// Initialisation des mutex
for (int i = 0; i < 5; i++) {
    pthread_mutex_init(&forks[i], NULL);
}

// Création des threads pour chaque philosophe
for (int i = 0; i < 5; i++) {
    pthread_create(&threads[i], NULL, routine, &philosophes[i]);
}

// Attendre que tous les threads aient terminé
for (int i = 0; i < 5; i++) {
    pthread_join(threads[i], NULL);
}

// Détruire les mutex pour les fourchettes
for (int i = 0; i < 5; i++) {
    pthread_mutex_destroy(&forks[i]);
}
```

En résumé

- `pthread_mutex_init` initialise un mutex (pour une fourchette).
- `pthread_create` crée un thread pour chaque philosophe.
- `pthread_join` attend que chaque thread ait terminé.

- `pthread_mutex_destroy` détruit les mutex une fois qu'ils ne sont plus utilisés.

Si tu as des questions ou besoin de plus de détails sur l'une de ces fonctions, n'hésite pas à demander ! 😊