fonctions Philosophers

Voici une explication détaillée pour chaque fonction que tu dois utiliser dans ton projet, avec leurs rôles, les paramètres qu'elles prennent et leurs valeurs de retour.

---

1. `usleep`

Description :
Met en pause l'exécution du thread appelant pendant un certain nombre de microsecondes.

Prototype :

```c
int usleep(useconds_t usec);
```

Paramètres :

- `usec` : Nombre de microsecondes à attendre (1 000 000 µs = 1 seconde).

Valeur de retour :

- `0` si la fonction réussit.
- `-1` en cas d'erreur (par exemple, si `usec` dépasse la limite ou si l'appel est interrompu par un signal).

Exemple :
Pause de 500 millisecondes :

```c
usleep(500000); // Attendre 500 000 microsecondes (0,5 seconde)
```

---

2. `gettimeofday`

Description :
Récupère l'heure actuelle (temps en secondes et microsecondes depuis l'époque Unix, 1er janvier 1970).

Prototype :

```c
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Paramètres :

- `tv` : Pointeur vers une structure `timeval` où la fonction stockera l'heure actuelle.
  - `tv->tv_sec` : secondes.
  - `tv->tv_usec` : microsecondes.
- `tz` : Optionnel, lié au fuseau horaire. Mets-le à `NULL` car il est obsolète.

Valeur de retour :

- `0` si la fonction réussit.
- `-1` en cas d'erreur.

Exemple :

```c
struct timeval tv;
gettimeofday(&tv, NULL);
printf("Secondes : %ld, Microsecondes : %ld\n", tv.tv_sec, tv.tv_usec);
```

---

3. `pthread_create`

Description :
Crée un nouveau thread et exécute une fonction dans ce thread.

Prototype :

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

Paramètres :

- `thread` : Pointeur vers un `pthread_t` pour stocker l'identifiant du thread créé.
- `attr` : Attributs du thread. Mets `NULL` pour utiliser les valeurs par défaut.
- `start_routine` : Fonction que le thread exécutera. Elle doit avoir la signature `void *function(void *arg)`.
- `arg` : Argument à passer à la fonction `start_routine`.

Valeur de retour :

- `0` si le thread a été créé avec succès.
- **Erreur (`> 0`)** en cas d'échec.

Exemple :

```c
pthread_t thread;
```

```
pthread_create(&thread, NULL, routine, &arg);
```

---

4. `pthread_detach`

Description :
Détache un thread, ce qui signifie que les ressources associées au thread seront automatiquement libérées lorsqu'il se termine, sans nécessiter un appel à `pthread_join`.

Prototype :

```c
int pthread_detach(pthread_t thread);
```

Paramètres :

- `thread` : Identifiant du thread à détacher.

Valeur de retour :

- `0` si l'opération réussit.
- **Erreur (`> 0`)** si elle échoue.

Exemple :

```c
pthread_detach(thread); // Libère automatiquement les ressources du thread
```

---

5. `pthread_join`

Description :
Attend qu'un thread spécifique se termine et récupère sa valeur de retour.

Prototype :

```c
int pthread_join(pthread_t thread, void **retval);
```

Paramètres :

- `thread` : Identifiant du thread à attendre.
- `retval` : Adresse où stocker la valeur de retour du thread. Mets `NULL` si tu n'en as pas besoin.

Valeur de retour :

- `0` si l'opération réussit.

- **Erreur (`> 0`)** si elle échoue.

Exemple :

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

---

6. `pthread_mutex_init`

Description :
Initialise un mutex pour synchroniser l'accès à une ressource partagée.

Prototype :

```c
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
```

Paramètres :

- `mutex` : Pointeur vers le mutex à initialiser.
- `attr` : Attributs du mutex (mets `NULL` pour les valeurs par défaut).

Valeur de retour :

- `0` si l'initialisation réussit.
- **Erreur (`> 0`)** si elle échoue.

Exemple :

```c
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
```

---

7. `pthread_mutex_destroy`

Description :
Détruit un mutex et libère ses ressources.

Prototype :

```c
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Paramètres :

- `mutex` : Pointeur vers le mutex à détruire.

Valeur de retour :

- `0` si l'opération réussit.
- **Erreur (`> 0`)** si elle échoue (par exemple, si le mutex est verrouillé).

Exemple :

```c
pthread_mutex_destroy(&mutex);
```

---

8. `pthread_mutex_lock`

Description :
Verrouille un mutex. Si le mutex est déjà verrouillé, le thread appelant est bloqué jusqu'à ce qu'il puisse acquérir le verrou.

Prototype :

```c
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Paramètres :

- `mutex` : Pointeur vers le mutex à verrouiller.

Valeur de retour :

- `0` si le verrouillage réussit.
- **Erreur (`> 0`)** si elle échoue.

Exemple :

```c
pthread_mutex_lock(&mutex); // Accès exclusif à une ressource
```

---

9. `pthread_mutex_unlock`

Description :
Déverrouille un mutex précédemment verrouillé par le thread appelant.

Prototype :

```c
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Paramètres :

- `mutex` : Pointeur vers le mutex à déverrouiller.

Valeur de retour :

- `0` si le déverrouillage réussit.
- **Erreur (`> 0`)** si elle échoue.

Exemple :

```c
pthread_mutex_unlock(&mutex); // Libère le verrou sur une ressource
```

Résumé

Fonction	Rôle	Retour (succès)	Retour (échec)
`usleep`	Pause en microsecondes	`0`	`-1`
`gettimeofday`	Récupère l'heure actuelle	`0`	`-1`
`pthread_create`	Crée un thread	`0`	> `0`
`pthread_detach`	Détache un thread	`0`	> `0`
`pthread_join`	Attend la fin d'un thread	`0`	> `0`
`pthread_mutex_init`	Initialise un mutex	`0`	> `0`
`pthread_mutex_destroy`	Détruit un mutex	`0`	> `0`
`pthread_mutex_lock`	Verrouille un mutex	`0`	> `0`
`pthread_mutex_unlock`	Déverrouille un mutex	`0`	> `0`

Si tu as des questions ou besoin d'un exemple plus détaillé pour une fonction spécifique, fais-le-moi savoir ! 😊