

# pthread\_create()

La fonction `pthread_create()` est utilisée en C pour créer un nouveau thread dans un programme multi-thread.

Voici la syntaxe générale de la fonction `pthread_create()` :

```
c Copy code  
  
#include <pthread.h>  
  
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,  
                  void *(*start_routine)(void *), void *arg);
```

Les arguments de la fonction `pthread_create()` sont :

**thread** : un pointeur vers un objet de type `pthread_t` qui contiendra l'identifiant du thread créé.

**attr** : un pointeur vers un objet de type `pthread_attr_t` qui contient les attributs du thread. Si vous voulez utiliser les attributs par défaut, vous pouvez passer `NULL` à cet argument.

**start\_routine** : un pointeur vers la fonction qui sera exécutée par le thread.

**arg** : un pointeur générique qui sera passé à la fonction `start_routine()`.

La fonction `pthread_create()` crée un nouveau thread et stocke son identifiant dans l'objet pointé par `thread`.

Le nouveau thread commence à s'exécuter en appelant la fonction `start_routine()` avec l'argument `arg`.

Voici un exemple simple d'utilisation de la fonction `pthread_create()` :

```
c Copy code

#include <stdio.h>
#include <pthread.h>

void *fonction_thread(void *arg)
{
    printf("Le thread a été créé avec succès !\n");
    return NULL;
}

int main()
{
    pthread_t thread;
    int ret;

    ret = pthread_create(&thread, NULL, fonction_thread, NULL);

    if (ret != 0) {
        printf("Erreur lors de la création du thread.\n");
        return 1;
    }

    printf("Le thread a été créé avec l'identifiant %lu.\n", thread);

    pthread_join(thread, NULL);

    return 0;
}
```

Dans cet exemple, la fonction `fonction_thread()` sera exécutée par le thread créé par la fonction `pthread_create()`.


La fonction `pthread_join()` est utilisée pour attendre la fin de l'exécution du thread avant que le programme ne se termine.

# pthread\_detach

La fonction `pthread_detach()` est utilisée en C pour détacher un thread. Cela signifie que les ressources associées au thread sont libérées automatiquement dès que le thread se termine. Cela permet de libérer la mémoire utilisée par le thread et d'empêcher toute fuite de mémoire.

Voici la syntaxe de la fonction `pthread_detach()` :

c

 Copy code

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

L'argument de la fonction `pthread_detach()` est un objet de type `pthread_t` qui représente le thread à détacher.

Voici un exemple d'utilisation de la fonction `pthread_detach()` :

```
c Copy code

#include <stdio.h>
#include <pthread.h>

void *fonction_thread(void *arg)
{
    printf("Le thread a été créé avec succès !\n");
    pthread_detach(pthread_self()); // Détacher le thread courant
    return NULL;
}

int main()
{
    pthread_t thread;
    int ret;

    ret = pthread_create(&thread, NULL, fonction_thread, NULL);

    if (ret != 0) {
        printf("Erreur lors de la création du thread.\n");
        return 1;
    }

    printf("Le thread a été créé avec l'identifiant %lu.\n", thread);

    pthread_join(thread, NULL);

    return 0;
}
```


Dans cet exemple, la fonction `pthread_detach()` est appelée dans la fonction `fonction_thread()` pour détacher le thread courant. Cela signifie que lorsque le thread se termine, les ressources associées à celui-ci sont automatiquement libérées, sans nécessiter l'appel à la fonction `pthread_join()`.

# pthread\_join

La fonction `pthread_join()` est utilisée en C pour attendre la fin de l'exécution d'un thread et récupérer sa valeur de retour.

Voici la syntaxe de la fonction `pthread_join()` :

c

 Copy code

```
#include <pthread.h>

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

Les arguments de la fonction `pthread_join()` sont :

**thread** : un objet de type `pthread_t` qui représente le thread à attendre.

**retval** : un pointeur générique qui sera utilisé pour stocker la valeur de retour du thread.

La fonction `pthread_join()` bloque l'exécution du thread appelant jusqu'à ce que le thread spécifié par l'argument `thread` se termine. Si le thread s'est terminé avant l'appel à `pthread_join()`, la fonction retourne immédiatement. Si le thread est encore en cours d'exécution, `pthread_join()` attend sa terminaison.

La valeur de retour du thread est stockée dans l'objet pointé par `retval`. Si vous ne voulez pas récupérer la valeur de retour du thread, vous pouvez passer `NULL` à cet argument.

Voici un exemple d'utilisation de la fonction `pthread_join()` :

```
#include <stdio.h>
#include <pthread.h>

void *fonction_thread(void *arg)
{
    printf("Le thread a été créé avec succès !\n");
    return NULL;
}

int main()
{
    pthread_t thread;
    int ret;

    ret = pthread_create(&thread, NULL, fonction_thread, NULL);

    if (ret != 0) {
        printf("Erreur lors de la création du thread.\n");
        return 1;
    }

    printf("Le thread a été créé avec l'identifiant %lu.\n", thread);

    ret = pthread_join(thread, NULL);

    if (ret != 0) {
        printf("Erreur lors de l'attente de la fin du thread.\n");
        return 1;
    }

    printf("Le thread s'est terminé avec succès !\n");

    return 0;
}
```

 Regenerate response

Dans cet exemple, la fonction `pthread_join()` est utilisée pour attendre que le thread créé par la fonction `pthread_create()` se termine. La fonction `pthread_join()` est appelée avec l'identifiant du thread à attendre et `NULL` comme deuxième argument car nous n'avons pas besoin de la valeur de retour du thread.

# pthread\_mutex\_init

La fonction `pthread_mutex_init()` est utilisée en C pour initialiser un objet de verrouillage (mutex) qui est utilisé pour synchroniser l'accès à des ressources partagées entre plusieurs threads. Cette fonction doit être appelée avant d'utiliser un objet de verrouillage.

Voici la syntaxe de la fonction `pthread_mutex_init()` :

```
c Copy code  
  
#include <pthread.h>  
  
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
```

Les arguments de la fonction `pthread_mutex_init()` sont :

**mutex** : un pointeur vers l'objet de verrouillage à initialiser.

**attr** : un pointeur vers un objet de type `pthread_mutexattr_t` qui contient les attributs du verrouillage. Si vous ne souhaitez pas spécifier d'attributs particuliers, vous pouvez passer `NULL`.

```

#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex;

void *fonction_thread(void *arg)
{
    // Verrouiller l'accès aux ressources partagées
    pthread_mutex_lock(&mutex);

    // Section critique
    printf("Le thread a accès aux ressources partagées !\n");

    // Déverrouiller l'accès aux ressources partagées
    pthread_mutex_unlock(&mutex);

    return NULL;
}

int main()
{
    pthread_t thread1, thread2;
    int ret;

    // Initialiser l'objet de verrouillage
    ret = pthread_mutex_init(&mutex, NULL);

    if (ret != 0) {
        printf("Erreur lors de l'initialisation du mutex.\n");
        return 1;
    }

    // Créer les threads
    ret = pthread_create(&thread1, NULL, fonction_thread, NULL);
    if (ret != 0) {
        printf("Erreur lors de la création du thread 1.\n");
        return 1;
    }

    ret = pthread_create(&thread2, NULL, fonction_thread, NULL);
    if (ret != 0) {
        printf("Erreur lors de la création du thread 2.\n");
        return 1;
    }

    // Attendre la fin des threads
    ret = pthread_join(thread1, NULL);
    if (ret != 0) {
        printf("Erreur lors de l'attente de la fin du thread 1.\n");
        return 1;
    }

    ret = pthread_join(thread2, NULL);
    if (ret != 0) {
        printf("Erreur lors de l'attente de la fin du thread 2.\n");
        return 1;
    }

    // Détruire l'objet de verrouillage
    ret = pthread_mutex_destroy(&mutex);
    if (ret != 0) {
        printf("Erreur lors de la destruction du mutex.\n");
        return 1;
    }

    return 0;
}

```

Voici un exemple  
d'utilisation de la fonction  
pthread\_mutex\_init() :



Dans cet exemple, la fonction `pthread_mutex_init()` est utilisée pour initialiser l'objet de verrouillage mutex. Les threads créés par la fonction `pthread_create()` utilisent ce mutex pour synchroniser leur accès à des ressources partagées.

Le mutex est verrouillé à l'aide de la fonction `pthread_mutex_lock()` et déverrouillé à l'aide de la fonction `pthread_mutex_unlock()`.

# pthread\_mutex\_destroy

La fonction `pthread_mutex_destroy()` est utilisée en C pour détruire un objet de verrouillage (mutex) qui a été initialisé avec la fonction `pthread_mutex_init()`. Cette fonction doit être appelée lorsque vous n'avez plus besoin d'utiliser l'objet de verrouillage.

Voici la syntaxe de la fonction `pthread_mutex_destroy()` :

```
c Copy code

#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

L'argument de la fonction `pthread_mutex_destroy()` est :

**mutex** : un pointeur vers l'objet de verrouillage à détruire.

Voici un exemple d'utilisation de la fonction `pthread_mutex_destroy()` :

```
c Copy code

#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex;

void *fonction_thread(void *arg)
{
    // Verrouiller l'accès aux ressources partagées
    pthread_mutex_lock(&mutex);

    // Section critique
    printf("Le thread a accès aux ressources partagées !\n");

    // Déverrouiller l'accès aux ressources partagées
    pthread_mutex_unlock(&mutex);

    return NULL;
}
```

```
int main()
{
    pthread_t thread1, thread2;
    int ret;

    // Initialiser l'objet de verrouillage
    ret = pthread_mutex_init(&mutex, NULL);

    if (ret != 0) {
        printf("Erreur lors de l'initialisation du mutex.\n");
        return 1;
    }

    // Créer les threads
    ret = pthread_create(&thread1, NULL, fonction_thread, NULL);
    if (ret != 0) {
        printf("Erreur lors de la création du thread 1.\n");
        return 1;
    }

    ret = pthread_create(&thread2, NULL, fonction_thread, NULL);
    if (ret != 0) {
        printf("Erreur lors de la création du thread 2.\n");
        return 1;
    }

    // Attendre la fin des threads
    ret = pthread_join(thread1, NULL);
    if (ret != 0) {
        printf("Erreur lors de l'attente de la fin du thread 1.\n");
        return 1;
    }

    ret = pthread_join(thread2, NULL);
    if (ret != 0) {
        printf("Erreur lors de l'attente de la fin du thread 2.\n");
        return 1;
    }

    // Détruire l'objet de verrouillage
    ret = pthread_mutex_destroy(&mutex);
    if (ret != 0) {
        printf("Erreur lors de la destruction du mutex.\n");
        return 1;
    }

    return 0;
}
```


Dans cet exemple, la fonction `pthread_mutex_destroy()` est utilisée pour détruire l'objet de verrouillage mutex. Cette fonction doit être appelée lorsque vous n'avez plus besoin d'utiliser le mutex.

# pthread\_mutex\_lock

La fonction `pthread_mutex_lock()` est utilisée en C pour verrouiller un objet de verrouillage (mutex). Elle permet de bloquer l'exécution d'un thread tant que le verrou n'est pas disponible.

Voici la syntaxe de la fonction `pthread_mutex_lock()` :

c

 Copy code

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
```

L'argument de la fonction `pthread_mutex_lock()` est :

**mutex** : un pointeur vers l'objet de verrouillage à verrouiller.

Voici un exemple d'utilisation de la fonction `pthread_mutex_lock()` :

```
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex;

void *fonction_thread(void *arg)
{
    // Verrouiller l'accès aux ressources partagées
    pthread_mutex_lock(&mutex);

    // Section critique
    printf("Le thread a accès aux ressources partagées !\n");

    // Déverrouiller l'accès aux ressources partagées
    pthread_mutex_unlock(&mutex);

    return NULL;
}
```

```
int main()
{
    pthread_t thread1, thread2;
    int ret;

    // Initialiser l'objet de verrouillage
    ret = pthread_mutex_init(&mutex, NULL);

    if (ret != 0) {
        printf("Erreur lors de l'initialisation du mutex.\n");
        return 1;
    }

    // Créer les threads
    ret = pthread_create(&thread1, NULL, fonction_thread, NULL);
    if (ret != 0) {
        printf("Erreur lors de la création du thread 1.\n");
        return 1;
    }

    ret = pthread_create(&thread2, NULL, fonction_thread, NULL);
    if (ret != 0) {
        printf("Erreur lors de la création du thread 2.\n");
        return 1;
    }

    // Attendre la fin des threads
    ret = pthread_join(thread1, NULL);
    if (ret != 0) {
        printf("Erreur lors de l'attente de la fin du thread 1.\n");
        return 1;
    }

    ret = pthread_join(thread2, NULL);
    if (ret != 0) {
        printf("Erreur lors de l'attente de la fin du thread 2.\n");
        return 1;
    }

    // Détruire l'objet de verrouillage
    ret = pthread_mutex_destroy(&mutex);
    if (ret != 0) {
        printf("Erreur lors de la destruction du mutex.\n");
        return 1;
    }

    return 0;
}
```

Dans cet exemple, la fonction `pthread_mutex_lock()` est utilisée pour verrouiller l'objet de verrouillage mutex avant d'accéder aux ressources partagées dans la section critique. L'appel à cette fonction bloque l'exécution du thread tant que le mutex n'est pas disponible. Le mutex sera déverrouillé lorsque le thread aura terminé l'accès aux ressources partagées et aura appelé la fonction `pthread_mutex_unlock()`.

# pthread\_mutex\_unlock

La fonction `pthread_mutex_unlock()` est utilisée en C pour déverrouiller un objet de verrouillage (mutex). Elle permet de libérer le verrou pour permettre à d'autres threads d'y accéder.

Voici la syntaxe de la fonction `pthread_mutex_unlock()` :

c

 Copy code

```
#include <pthread.h>

int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

L'argument de la fonction `pthread_mutex_unlock()` est :

**mutex** : un pointeur vers l'objet de verrouillage à déverrouiller.

Voici un exemple d'utilisation de la fonction `pthread_mutex_unlock()` :

```
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex;

void *fonction_thread(void *arg)
{
    // Verrouiller l'accès aux ressources partagées
    pthread_mutex_lock(&mutex);

    // Section critique
    printf("Le thread a accès aux ressources partagées !\n");

    // Déverrouiller l'accès aux ressources partagées
    pthread_mutex_unlock(&mutex);

    return NULL;
}

int main()
{
    pthread_t thread1, thread2;
    int ret;

    // Initialiser l'objet de verrouillage
    ret = pthread_mutex_init(&mutex, NULL);

    if (ret != 0) {
        printf("Erreur lors de l'initialisation du mutex.\n");
        return 1;
    }
}
```



```

// Créer les threads
ret = pthread_create(&thread1, NULL, fonction_thread, NULL);
if (ret != 0) {
    printf("Erreur lors de la création du thread 1.\n");
    return 1;
}

ret = pthread_create(&thread2, NULL, fonction_thread, NULL);
if (ret != 0) {
    printf("Erreur lors de la création du thread 2.\n");
    return 1;
}

// Attendre la fin des threads
ret = pthread_join(thread1, NULL);
if (ret != 0) {
    printf("Erreur lors de l'attente de la fin du thread 1.\n");
    return 1;
}

ret = pthread_join(thread2, NULL);
if (ret != 0) {
    printf("Erreur lors de l'attente de la fin du thread 2.\n");
    return 1;
}

// Détruire l'objet de verrouillage
ret = pthread_mutex_destroy(&mutex);
if (ret != 0) {
    printf("Erreur lors de la destruction du mutex.\n");
    return 1;
}

return 0;
}

```

🔄 Regenerate response


Dans cet exemple, la fonction `pthread_mutex_unlock()` est utilisée pour déverrouiller l'objet de verrouillage mutex après que le thread a terminé l'accès aux ressources partagées dans la section critique. Le mutex est déverrouillé pour permettre à d'autres threads d'y accéder.

# gettimeofday

La fonction `gettimeofday()` est une fonction en C qui permet d'obtenir le temps courant en microsecondes depuis l'Epoch UNIX. Elle est définie dans la bibliothèque `sys/time.h`.

Voici la syntaxe de la fonction `gettimeofday()` :

c

 Copy code

```
#include <sys/time.h>

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


Les arguments de la fonction `gettimeofday()` sont :

**tv** : un pointeur vers une structure `timeval` qui contiendra la valeur du temps.

**tz** : un pointeur vers une structure `timezone` qui peut être ignorée en passant la valeur `NULL`.

La structure `timeval` est définie comme suit :

c

 Copy code

```
struct timeval {
    time_t      tv_sec;        /* secondes */
    suseconds_t tv_usec;      /* microsecondes */
};
```

La fonction `gettimeofday()` remplit la structure `timeval` avec la valeur du temps courant. Les secondes sont stockées dans le champ `tv_sec` et les microsecondes dans le champ `tv_usec`.

Voici un exemple d'utilisation de la fonction `gettimeofday()` :

```
#include <stdio.h>
#include <sys/time.h>

int main()
{
    struct timeval tv;
    int ret;

    ret = gettimeofday(&tv, NULL);
    if (ret != 0) {
        printf("Erreur lors de l'appel à gettimeofday().\n");
        return 1;
    }
}
```

```
    printf("Temps courant en microsecondes depuis l'Epoch UNIX : %ld\n", tv.tv_sec * 1000000L + tv.tv_usec);
    return (0);
}
```

Dans cet exemple, la fonction `gettimeofday()` est utilisée pour obtenir le temps courant en microsecondes depuis l'Epoch UNIX. La valeur du temps est stockée dans la structure `timeval tv`, qui est ensuite affichée à l'aide de la fonction `printf()`.