

Chapitre V : Threads

Cours Système Exploitation II

1 Linfo

ISIMM

2022/2023

Introduction

- Programme multitâche → de lancer plusieurs parties de son code en même temps.
- À chaque partie du code sera associé un sous-processus pour permettre l'exécution en parallèle.
- Exemple : ouvrir plusieurs onglets d'un navigateur : un onglet pour lire un article en ligne, un autre pour regarder youtube et un autre pour utiliser ChaGPT.
- Processus coûte cher au lancement: espace mémoire, la commutation, la communication inter-processus (échange d'informations) seront lourdes à gérer



Introduction

- Peut-on avoir un sous-processus qui permettrait de lancer une partie du code d'une application sans qu'il soit onéreux?
- **Les threads qui sont appelés aussi « processus légers ».**



Processus vs Threads

Processus A

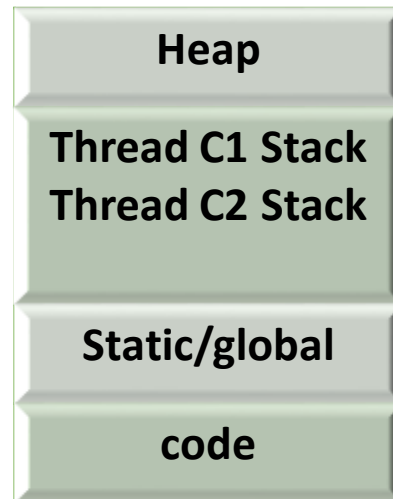


Processus B

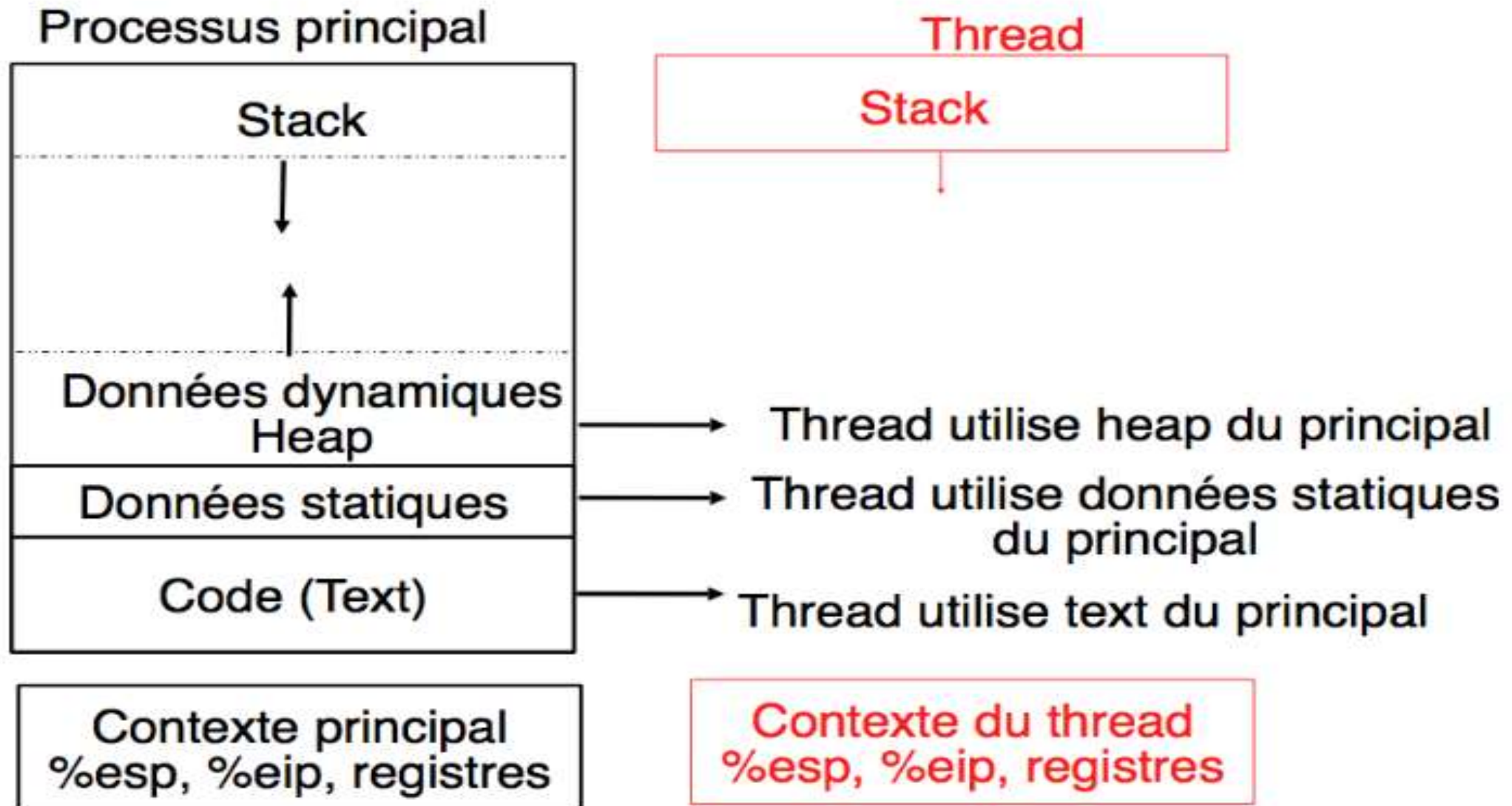


**Processus C avec
2 threads fait le
même travail que
processus A et B**

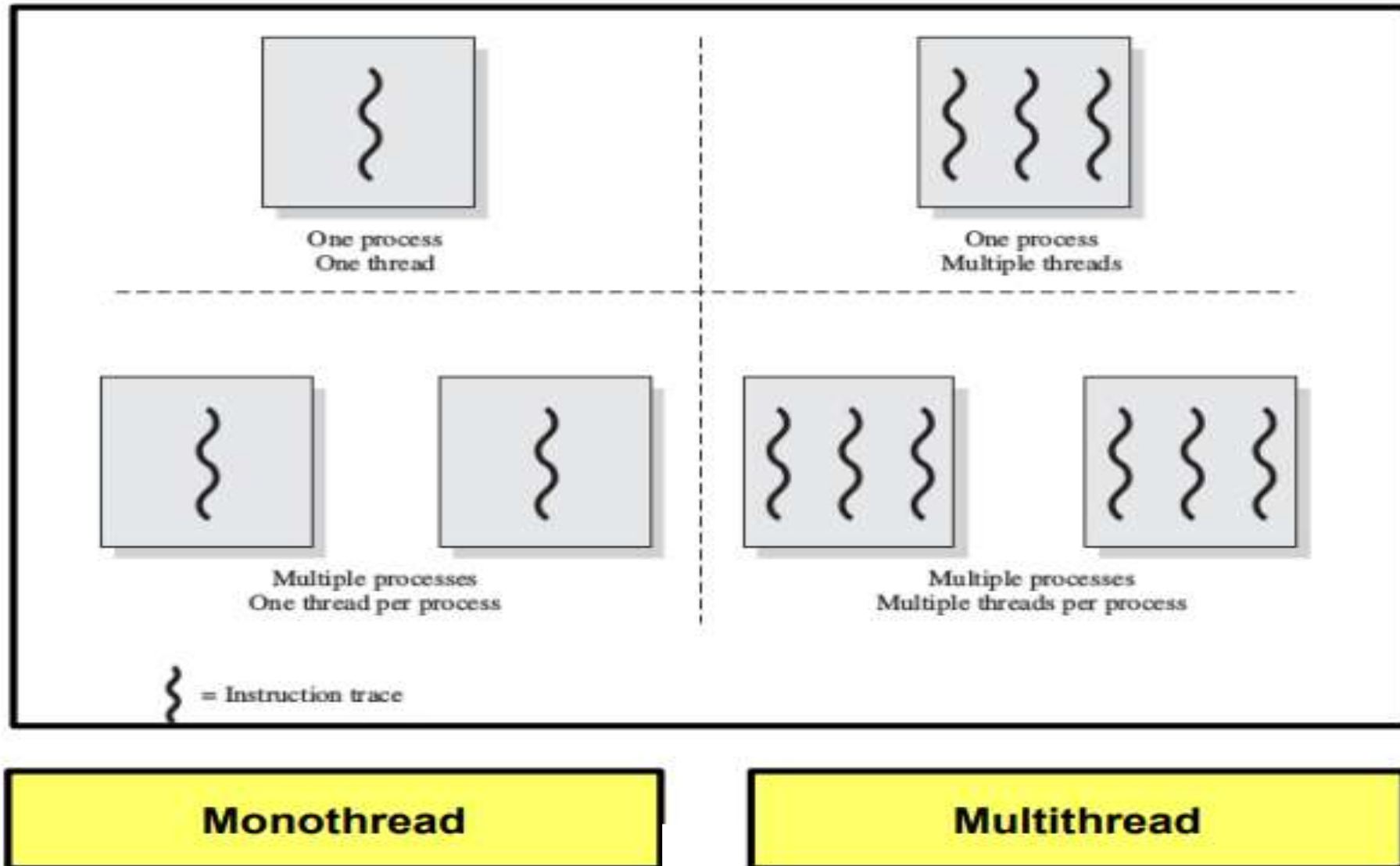
Processus C



Processus vs Threads



Monothread vs Multithread



Définition : Thread

- Un thread est un sous-ensemble d'un processus, partageant son espace mémoire et ses variables.
- Plus rapide que le processus
- A chaque thread est associé des unités propres à lui:
 - Sa pile (pour gérer les instructions à exécuter par le thread),
 - Le masque de signaux (les signaux que le thread doit répondre),
 - Sa priorité d'exécution (dans la file d'attente),
 - Des données privées.

Avantages / Inconvénients



- Partage de la mémoire : mécanisme rapide de communication inter-thread



- plus léger : moins de données système à recopier



- plus rapide : le context-switch est plus facile



- Les threads utilisent les mêmes copies des bibliothèques : les bibliothèques doivent être "MT-safe":



- Il faut gérer la synchronisation (mutex, sémaphores...):

- En cas de mauvaise synchronisation :



- Comportement "aléatoire" : bugs, segfaults, exploitations...

- Interblocage

Création d'un thread

- On déclare d'abord une instance de pthread :

```
pthread_t ta; // ta une instance de pthread
```

- Pour créer un thread, il faudra utiliser la fonction « **pthread_create** ». La signature d'une telle fonction est comme suit :

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

Création et terminaison d'un thread

- Cette fonction retourne « 0 » si le thread a été créé.

Un pointeur vers une instance de « pthread ».
Ce pointeur contient l'identificateur du thread créé

« attr » contient les attributs du thread à créer. On passe la valeur « NULL » pour initialiser les attributs avec les valeurs par défaut.

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

« arg » est l'argument que nous allons passer à la fonction « start_routine ».

Une fonction que le thread va exécuter. Cette fonction retourne un « void* » et accepte un argument du type « void* ».

Identifiant d'un pthread

- Un pthread n'a pas de PID propre (ils partagent tous le même PID, celui du processus contenant les threads)
- Chaque thread possède un identifiant unique de type `pthread_t` (équivalent du `pid_t`, c'est une structure opaque)
- `Pthread_t pthread_self()` : retourne l'ID du thread (équivalent à `getpid()`)
- `Pthread_equal(pthread_t a, pthread_t b)` permet de comparer deux identifiants

Attente d'un thread

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

`retval` : un pointeur sur une variable (contenant un pointeur), où le code retour sera copié.

- ▶ `NULL` : ignoré
- ▶ thread "annulé" : `PTHREAD_CANCELED`

Terminaison d'un thread

```
int pthread_exit(void *retval);
```

Arrête (termine) le thread courant.

`retval` sera la valeur retour (récupérée par `pthread_join`)

Example 1

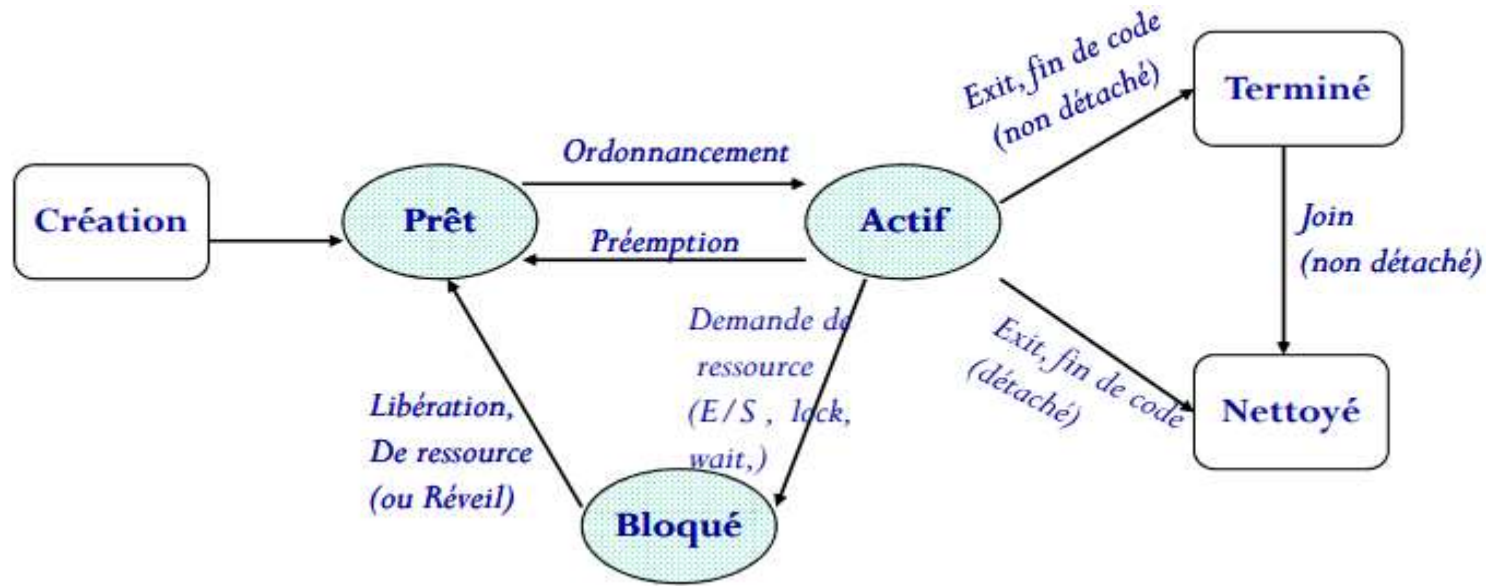
thread-ex1.c

```
#include <pthread.h>
void *hello( void *arg ) {
    int *id = (int*)arg;
    printf("%d : hello world \n", *id);
    pthread_exit(NULL);
}
int main (int argc, char *argv[ ]) {
    pthread_t thread[3];
    int id[3]={1,2,3};
    int i;

    for (i=0;i<3;i++) {
        printf("Crée thread %d\n",i);
        pthread_create(&thread[i], NULL,
                      hello, (void *)&id[i]);
    }
    pthread_exit(NULL);
}
```

```
$ gcc -pthread -o thread-ex1 thread-ex1.c
$ ./thread-exemple1
Crée thread 1
Crée thread 2
Crée thread 3
1 : hello world
2 : hello world
3 : hello world
$
```

Cycle de vie d'un thread



Si on ne veut pas avoir à gérer la fin d'un thread, on peut le "détacher".

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

- ▶ La structure sera détruite à la terminaison du thread
- ▶ Il ne sera pas possible de retrouver son pointeur retour avec `pthread_join`

Exemple 2 : en c++

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

using namespace std;

const int N=10;

void * writer_thread (void * arg) {
    int i;
    for (i = 0; i < N; i++)
        cout << "    Thread enfant.\n";
    return NULL;
}

int main (int argc, char *argv[]) {
    int i;
    pthread_t writer_id;

    pthread_create (&writer_id, NULL, writer_thread, NULL);
    for(i = 0; i < N; i++)
        cout << "Thread Parent\n";

    pthread_exit (NULL);
    return 0;
}
```

Thread « enfant » représenté par l'instance « writer_id ».

Sortie -1-	Sortie -2-
Thread enfant.	Thread Parent
Thread enfant.	Thread Parent
Thread enfant.	Thread enfant.
Thread enfant.	Thread enfant.
Thread enfant.	Thread enfant.
Thread enfant.	Thread enfant.
Thread enfant.	Thread enfant.
Thread enfant.	Thread enfant.
Thread Parent	Thread enfant.
Thread Parent	Thread enfant.
Thread Parent	Thread enfant.
Thread Parent	Thread enfant.
Thread Parent	Thread Parent
Thread Parent	Thread Parent
Thread Parent	Thread Parent
Thread Parent	Thread Parent
Thread Parent	Thread Parent
Thread Parent	Thread Parent
Thread enfant.	Thread Parent
Thread enfant.	

Thread parent représenté par la fonction « main »

Exemple 2 : en c++

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

using namespace std;

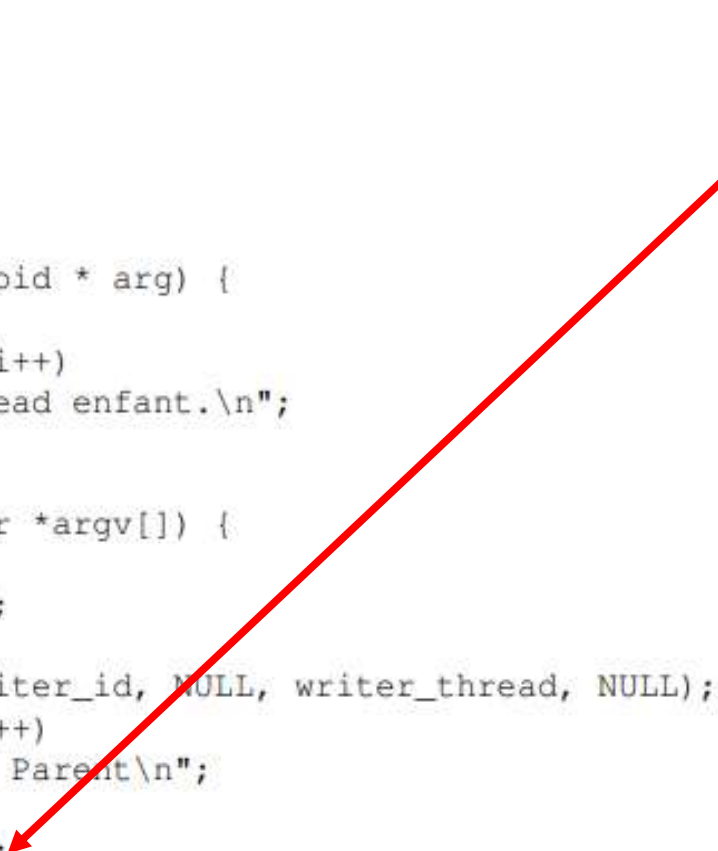
const int N=10;

void * writer_thread (void * arg) {
    int i;
    for (i = 0; i < N; i++)
        cout << "    Thread enfant.\n";
    return NULL;
}

int main (int argc, char *argv[]) {
    int i;
    pthread_t writer_id;

    pthread_create (&writer_id, NULL, writer_thread, NULL);
    for(i = 0; i < N; i++)
        cout << "Thread Parent\n";

    pthread_exit (NULL);
    return 0;
}
```



La fonction « pthread_exit(NULL) » sert à terminer un thread.

Il faudra utiliser cette fonction à la place de « exit » afin d'éviter d'arrêter tout le processus.

L'appel à une telle fonction à la fin de la fonction « main » est utile pour éviter que le programme arrête les autres threads en cours.

Exemple 2 : en c++

```
int main (int argc, char *argv[]) {
    int i;

    cout << "***** Debut de la fonction main *****"<<endl;

    // Un thread, une instance de pthread_t
    pthread_t writer_id;

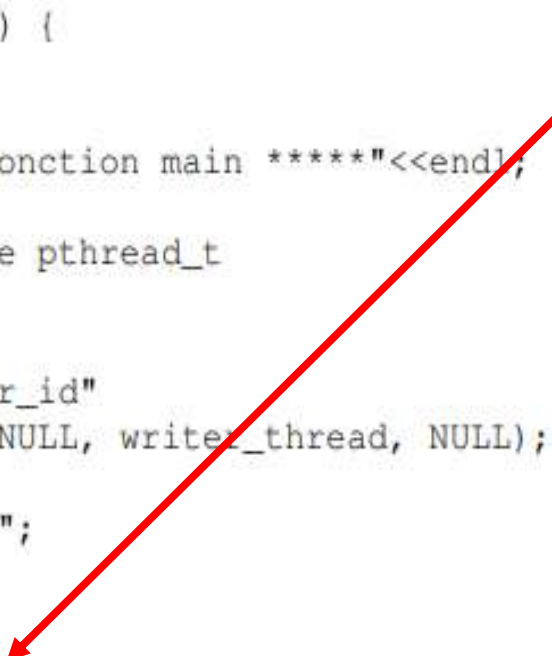
    // Création du thread "writer_id"
    pthread_create (&writer_id, NULL, writer_thread, NULL);
    for(i = 0; i < N; i++)
        cout << "Thread Parent\n";

    pthread_join(writer_id, NULL);

    cout << "***** Fin de la fonction main *****"<<endl;

    // Terminaison du thread
    pthread_exit (NULL);

    return 0;
}
```



En ajoutant la fonction «pthread_join», on met en attente le programme tant que le thread « writer_id » n'a pas terminé de s'exécuter.

- La fonction « pthread_join » c'est une des formes de synchronisation entre les threads.
- Par défaut dans le standard POSIX, tous les threads sont créés dans l'état « joignable » par opposé à l'état « détaché ».

Exemple 2 : en c++

- La fonction « pthread_detach » est utilisée pour détacher un thread. Quand ce thread prendra fin, l'espace mémoire occupé par celui-ci sera libéré.
- En le mettant dans un état détaché, nous n'avons pas besoin d'attendre qu'un autre thread fasse un appel à pthread_join.

```
int pthread_detach(pthread_t th);
```

Résumé des principales fonctions

Nom de la fonction	Rôle de la fonction
Pthread_create(pthread_t tid, ..., void* ((*fonction), void* arg))	Création d'un thread Par défaut, tous les threads ont la même priorité. Son démarrage dépend de sa priorité. On peut changer sa priorité.
Pthread_exit(void* etat)	Termine le thread en fournissant un code de retour, à la différence de exit qui termine le processus et tous ses threads.
Pthread_self(void)	Identification du numéro (tid) du thread courant, équivalent de getpid().
pthread_join (pthread_t tid, void ** etat)	Pour attendre la fin d'un thread dont on donne le tid.

Exercice

- Ecrire un programme qui crée deux threads pour exécuter chacun deux fonctions .
- Le premier thread affiche des étoiles et le second des dièses

Exercise

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

void *afficher_etoiles(void *arg) {
    int i;
    for (i = 0; i < 10; i++) {
        printf("* ");
    }
    printf("\n");
    pthread_exit(NULL);
}

void *afficher_dieses(void *arg) {
    int i;
    for (i = 0; i < 10; i++) {
        printf("# ");
    }
    printf("\n");
    pthread_exit(NULL);
}
```

Exercice

```
int main() {
    pthread_t thread1, thread2;
    int res1, res2;

    res1 = pthread_create(&thread1, NULL, afficher_etoiles, NULL);
    if (res1 != 0) {
        printf("Erreur lors de la création du thread 1\n");
        exit(EXIT_FAILURE);
    }

    res2 = pthread_create(&thread2, NULL, afficher_dieses, NULL);
    if (res2 != 0) {
        printf("Erreur lors de la création du thread 2\n");
        exit(EXIT_FAILURE);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```