

Programmation Système

SMI S6

Chapitre 8 : Les Threads

Mohammed Benattou

1. Introduction

Un **thread** (*Fil ou encore Fil d'exécution*) est une portion de code (*fonction, procédure, routine*) qui se s'exécute en parallèle au thread principal qui l'a crée (*main*).

Propriétés :

- ❑ Principe semblable à la fonction [**fork**](#) sans copie des ressources du processus père
- ❑ Programmation multitâche définitions des fonctions qui vont s'exécutés en même temps que le processus créateur.
- ❑ Permettre à un programme de réaliser plusieurs actions au même temps (simultanément)
- ❑ Un thread peut être considéré comme un processus allégé (on parle de processus léger). En comparaison des threads, un fork prend en moyenne 30 fois plus de temps à réaliser la même tâche

2. Création d'un Thread

La fonction **pthread_create()** permet la création d'un thread

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

- **thread** : correspond à l'identifiant du thread de type **pthread_t** est un type prédéfini (*sur Linux **unsigned long***), comme les processus ont leur propre identifiant
- **attr** : de type **pthread_attr_t** permettant de définir des attributs spécifiques pour chaque thread. Les attributs permettent de spécifier la taille de la pile, la priorité, la politique de planification, etc (par défaut mettre cet argument à NULL).
- **start_routine** : Chaque thread dispose d'une fonction à exécuter. Cet argument permet de transmettre un pointeur sur la fonction à exécuter.
- **arg** : argument que l'on peut passer à la fonction que le thread doit exécuter.

Valeur de retour :

- Si la création réussit, la fonction renvoie **0** et l'identifiant du thread nouvellement créé est stocké dans thread
- En cas d'erreur, la valeur **EAGAIN** (pas assez de ressources, ou nombre maximum de threads > **PTHREAD_THREADS_MAX**)

Exemple :

Threads	Programme Principale
<pre>#include<stdio.h> #include<stdlib.h> #include <sys/types.h> #include <unistd.h> #include <pthread.h> void *funct1() { int i; for(i=0; i<10; i++) { printf("Bonjour "); } } void *funct2() { int i; for(i=0; i<10; i++) { printf("Bonsoir "); } }</pre>	<pre>int main(){ pthread_t t1; pthread_t t2; pthread_create(&t1, NULL, funct1, NULL); pthread_create(&t2, NULL, funct2, NULL); return 0; }</pre>

Le programme principal crée deux Threads qui doivent afficher les messages **Bonjour** et **Bonsoir**.

A l'**exécution** rien ne se passe : ?

Aucun affichage en effet **trois processus s'exécutent en parallèle**

- main,
- threads t1
- threads t2

Il se peut que main exécute le return 0, avant d'exécuter les Thread t1 et t2

Solution1 :

On modifie le programme de tel manière que le processus main attend la fin de l'exécution des threads t1 et t2 par un sleep() ou un traitement particulier (boucle)

Threads	Programme Principale
<pre> #include<stdio.h> #include<stdlib.h> #include <sys/types.h> #include <unistd.h> #include <pthread.h> void *funct1() { int i; for(i=0; i<10; i++) { printf("Bonjour "); } } void *funct2() { int i; for(i=0; i<10; i++) { printf("Bonsoir "); } } </pre>	<pre> int main(){ pthread_t t1; pthread_t t2; pthread_create(&t1, NULL, funct1, NULL); pthread_create(&t2, NULL, funct2, NULL); sleep(1); return 0; } </pre>

t2; t1	t1; t2
<p> Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour </p>	<p> Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonjour Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir Bonsoir </p>

On remarque à l'exécution que le système peut commencer avec t1 ou t2

Combien de temps doit attendre le programme principal la fin de l'exécution d'un thread ?

En effet, l'exécution ne doit pas prendre le risque de terminer le programme complètement sans avoir pu exécuter les threads (exécution du main).

2. Attente d'un Thread

La fonction **pthread_join ()** permet au processus qui fait l'appel d'attendre que le Thread créé termine son exécution

int pthread_join (pthread_t thread, void ** thread_return);

Ses arguments sont dans l'ordre :

- Le thread à attendre **thread**
- La valeur de retour de la fonction du thread **thread**

L'appel de cette fonction met en pause l'exécution du thread appelant jusqu'au retour de la fonction correspondant au thread appelé

Valeur de retour :

- ✓ Si aucun problème n'a eu lieu, elle retourne **0(zéro)** et la valeur de retour du thread est passée à l'adresse indiquée (second argument) si elle est différente de NULL.
- ✓ En cas de problème, la fonction retourne une des valeurs suivantes :
 - **ESRCH** : aucun thread ne correspond à celui passé en argument.
 - **EINVAL** : le thread a été détaché ou un autre thread attend déjà la fin du même thread.
 - **EDEADLK** : le thread passé en argument correspond au thread appelant.

Exemple :

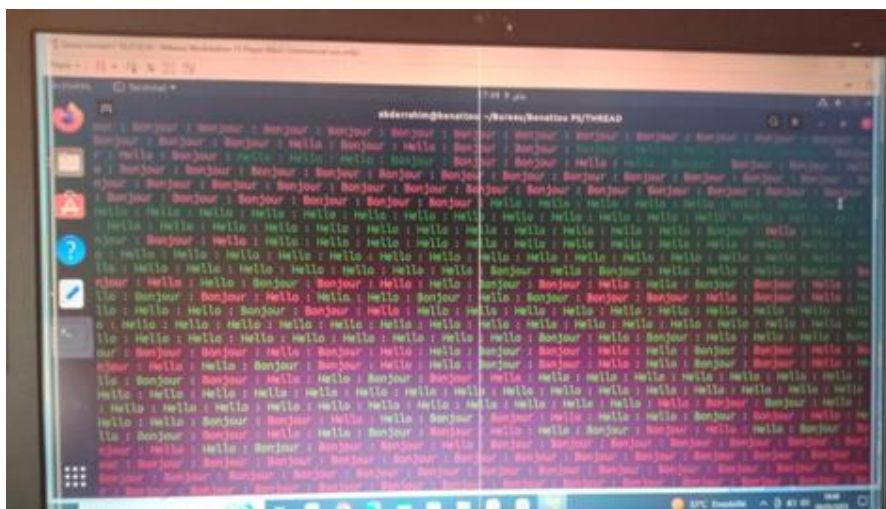
Dans l'exemple suivant, On arrête l'exécution du Thread courant avec la fonction :

void pthread_exit (void * retval)

Son seul argument est le retour de la fonction du thread appelant. Cet argument peut aussi être récupéré par la fonction **pthread_join()**

Threads	Programme Principale
<pre> #include<stdio.h> #include<stdlib.h> #include <sys/types.h> #include <unistd.h> #include <pthread.h> #define couleur(param) printf("\033[%dm",param) void *funct1(void *arg) { for(int i=0; i<10000; i++) { couleur(31); //afficher en rouge printf("Bonjour : "); } pthread_exit(NULL); } void *funct2(void *arg) {for(int i=0; i<10000; i++) { couleur(32); // afficher en vert printf("Hello : "); } pthread_exit(NULL); } </pre>	<pre> int main(){ pthread_t t1; pthread_t t2; pthread_create(&t1, NULL, funct1, NULL); pthread_create(&t2, NULL, funct2, NULL); pthread_join(t1,NULL); pthread_join(t2,NULL); return 0; } </pre>

Capture d'Exécution :

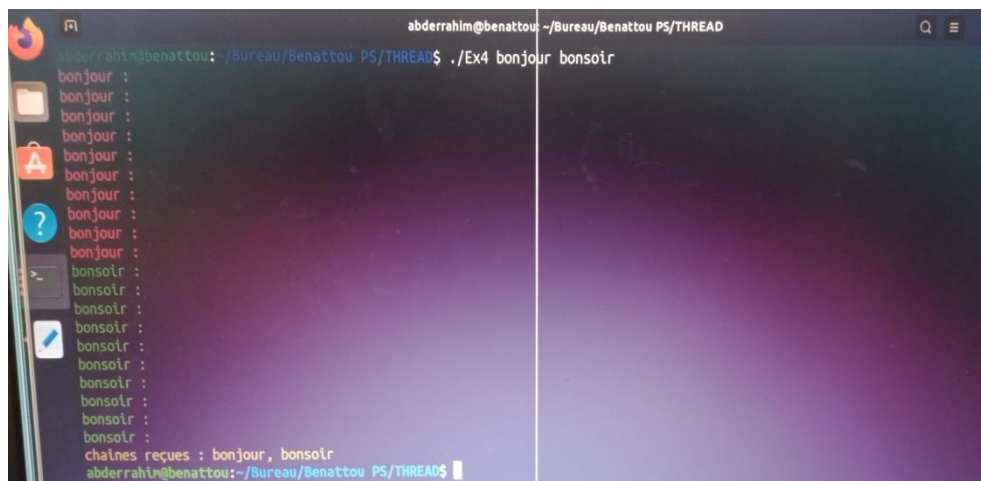


3. Exécution de Thread avec passage de Paramètre

Dans l'exemple suivant, on crée deux Thread avec la fonction **pthread_create()** en passant des arguments aux fonctions correspondant aux Thread créés. Dans cet exemple des chaînes de caractères sont passées en paramètres.

Threads avec Paramètre	Programme Principale
<pre>include<stdio.h> #include<stdlib.h> #include <sys/types.h> #include <unistd.h> #include <pthread.h> #define couleur(param) printf("\033[%dm",param) void *funct1(void *arg) { for(int i=0; i<10; i++) { couleur(31); //afficher en rouge printf("%s : \n", (char *) arg); } pthread_exit(arg); } void *funct2(void *arg) for(int i=0; i<10; i++) { couleur(32); // afficher en vert printf("%s : \n", (char *) arg); } pthread_exit(arg); }</pre>	<pre>int main(int argc, char *argv[]) { char *S1, *S2; pthread_t t1; pthread_t t2; pthread_create(&t1, NULL, funct1, argv[1]); pthread_create(&t2, NULL, funct2, argv[2]); //on récupère la valeur retournée par le thread pthread_join(t1, (void **) &S1); pthread_join(t2,(void **) &S2); //le programme principal affiche les chaînes reçues couleur(33); // afficher en vert printf("chaînes reçues : %s, %s \n", S1, S2); return 0; }</pre>

Capture d'Exécution :



```
abderrahim@benattou: ~/Bureau/Benattou PS/THREAD
abderrahim@benattou: ~/Bureau/Benattou PS/THREAD$ ./Ex4 bonjour bonsoir
bonjour :
bonjour :
bonjour :
bonjour :
bonjour :
bonjour :
bonjour :
bonjour :
bonjour :
bonsoir :
bonsoir :
bonsoir :
bonsoir :
bonsoir :
bonsoir :
bonsoir :
bonsoir :
bonsoir :
chaînes reçues : bonjour, bonsoir
abderrahim@benattou:~/Bureau/Benattou PS/THREAD$
```

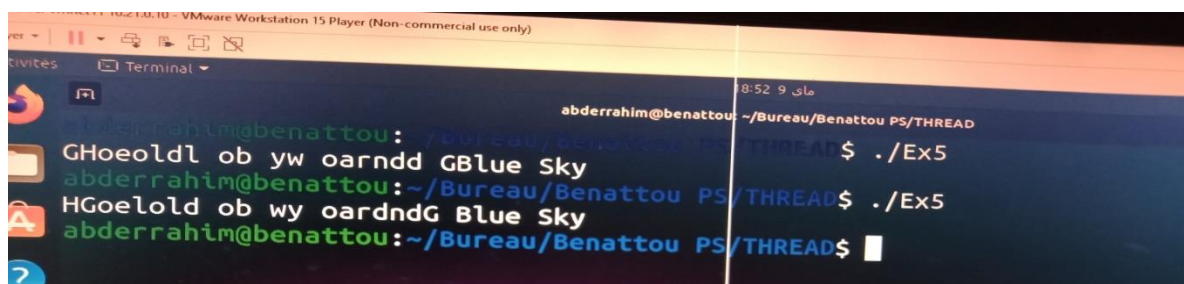
4. Thread et Exclusion Mutuelle

Le problème de l'exclusion mutuelle se pose lorsque deux Thread exécutent en même temps leurs sections critiques sans gestion de l'exclusion mutuelle.

Threads Sans gestion de l'exclusion Mutuelle	Programme Principale
<pre>#include<stdio.h> #include<stdlib.h> #include <sys/types.h> #include <unistd.h> #include <pthread.h> void *funct1(void *arg) { char *str ="Hello word"; for(int i=0; i<12; i++) { printf("%c", str[i]); usleep(1000); } } void *funct2(void *arg) { char *str ="Good by and Blue Sky"; for(int i=0; i<20; i++) { printf("%c", str[i]); usleep(1000); } }</pre>	<pre>int main(){ pthread_t t1; pthread_t t2; pthread_create(&t1, NULL, funct1, NULL); pthread_create(&t2, NULL, funct2, NULL); pthread_join(t1,NULL); pthread_join(t2,NULL); printf("\n"); return 0; }</pre>

Capture d'Exécution :

On remarque que les deux chaines sont écrites en chevauchant les caractères.



Fonctions de Gestion de l'Exclusion Mutuelle

Pour créer une variable pour la gestion de l'exclusion mutuelle (*mutex*), on doit déclarer une variable du type **pthread_mutex_t**

L'initialiser avec la constante **PTHREAD_MUTEX_INITIALIZER**

Exemple : **pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;**

En effet Un mutex n'a que deux états possibles :

a. verrouillé (blocage)

La fonction **pthread_mutex_lock()** permet de verrouiller l'accès à la section critique pour les autres threads

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

Valeur de retour :

- **0** en cas de succès
- en cas d'échec : **-EINVAL** : mutex non initialisé.
-EDEADLK : mutex déjà verrouillé par un thread différent

b. déverrouillé (déblocage)

La fonction **pthread_mutex_unlock ()** permet de déverrouiller le verrou de l'exclusion mutuelle passé en argument

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

Valeur de retour :

- **0** en cas de succès
- En cas d'échec : **-EINVAL** : mutex non initialisé
-EPERM : le thread n'a pas la main sur le mutex

Exemple : modification du programme précédent pour la prise en charge de l'exclusion mutuelle

Threads avec Gestion de l'Exclusion Mutuelle	Programme Principale
<pre> #include<stdio.h> #include<stdlib.h> #include <sys/types.h> #include <unistd.h> #include <pthread.h> pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; void *funct1(void *arg) {char *str =" Hello world"; pthread_mutex_lock(&mutex); for(int i=0; i<12; i++) {printf("%c", str[i]); usleep(1000); } pthread_mutex_unlock(&mutex) } void *funct2(void *arg) { char *str =" Good by and Blue Sky"; pthread_mutex_lock(&mutex); for(i=0; i<21; i++) {printf("%c", str[i]); usleep(1000);} pthread_mutex_unlock(&mutex); } </pre>	<pre> int main(){ pthread_t t1; pthread_t t2; pthread_create(&t1, NULL, funct1, NULL); pthread_create(&t2, NULL, funct2, NULL); pthread_join(t1,NULL); pthread_join(t2,NULL); /* Détruire le mutex */ pthread_mutex_destroy(&mutex); printf("\n"); return 0; } </pre>

Capture d'Exécution : On remarque que les deux chaines sont bien écrites.

```

abderrahim@benattou: ~/Bureau/Benattou PS/THREAD
abderrahim@benattou:~/Bureau/Benattou PS/THREAD$ ./Ex6
Good by and Blue Sky Hello world
abderrahim@benattou:~/Bureau/Benattou PS/THREAD$ ./Ex6
Hello world Good by and Blue Sky
abderrahim@benattou:~/Bureau/Benattou PS/THREAD$

```

5. Synchronisation avec condition

C'est un mécanisme de synchronisation qui permet à un thread de suspendre son exécution jusqu'à ce que une certaine condition soit vérifiée (principe semblable aux signaux et aux sémaphores).

Deux opérations sont fondamentales pour les conditions:

a. Signaler la condition (quand le prédicat devient vrai)

Fonction qui permet de signaler que la condition est remplie ce qui réveille alors le thread qui est en attente de cette condition.

int pthread_cond_signal(pthread_cond_t *cond)

b. Attendre la condition

Fonction qui permet de suspendre l'exécution du thread qui fait l'appel jusqu'à ce que un autre thread signale la condition (signale la fin de l'attente)

int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)

L'accès à la condition est protégé par l'exclusion mutuelle mutex

Initialisation de mutex et condition :

`pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;`

`pthread_cond_t cond = PTHREAD_COND_INITIALIZER;`

Exemple : dans l'exemple suivant on veut gérer l'affichage des deux chaines dans un ordre bien défini.

Threads avec Gestion de Condition	Programme Principale
<pre> #include<stdio.h> #include<stdlib.h> #include <sys/types.h> #include <unistd.h> #include <pthread.h> void *funct1(void *arg) { char *str =" Hello world"; for(int i=0; i<12; i++) { printf("%c", str[i]); usleep(1000); } pthread_cond_signal(&cond); /* Réveille*/ } void *funct2(void *arg) { int i; char *str =" Good by and Blue Sky"; pthread_cond_wait(&cond,&mutex) /*Attente*/ for(i=0; i<21; i++) { printf("%c", str[i]); usleep(1000); } } </pre>	<pre> // Initialisation de mutex et condition pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; pthread_cond_t cond = PTHREAD_COND_INITIALIZER; int main(){ pthread_t t1; pthread_t t2; pthread_create(&t1, NULL, funct1, NULL); pthread_create(&t2, NULL, funct2, NULL); pthread_join(t1,NULL); pthread_join(t2,NULL); /* Suppression */ pthread_mutex_destroy(&mutex); pthread_cond_destroy(&cond); printf("\n"); return 0; }} </pre>

Capture d'Exécution : On remarque que les deux chaines sont écrites dans l'ordre programmé.

```

abderrahim@benattou: ~/Bureau/Benattou PS/THREAD
abderrahim@benattou:~/Bureau/Benattou PS/THREAD$ ./Ex7
Hello world Good by and Blue Sky
abderrahim@benattou:~/Bureau/Benattou PS/THREAD$ ./Ex7
Hello world Good by and Blue Sky
abderrahim@benattou:~/Bureau/Benattou PS/THREAD$

```