Programmation concurrente
Mise en œuvre des threads en C

Définition

Généralement, pour chaque applicatif, le système réserve ce que l'on appelle un processus.

Dans la plupart des systèmes d'exploitation, chaque processus possède un espace d'adressage et un **thread** de contrôle unique, le **thread principal**. Du point de vue programmation, ce thread principal exécute le *main()*.

Le mot « *thread* » est un terme anglais qui peut se traduire par « *fil d'exécution* ». L'appellation de « *processus léger* » est également utilisée.

Avantage des threads par rapport aux processus
Facilité et rapidité de leur création (100 fois plus rapides en moyenne). Tous les threads d'un même processus partagent le même espace d'adressage, et donc toutes les variables. Cela évite donc l'allocation de tous ces espaces lors de la création.

Superposition de l'exécution des activités dans une même application (Importante accélération quant au fonctionnement de cette dernière.

La communication entre les threads est plus aisée que celle entre les processus, pour lesquels on doit utiliser des notions compliquées comme les tubes.

Compilation

Toutes les fonctions relatives aux *threads* sont incluses dans le fichier d'en-tête **<pthread.h>** et dans la bibliothèque **libpthread.a** (soit -lpthread à la compilation)

Exemple : Écrivez la ligne de commande qui vous permet de compiler votre programme sur les *threads* constitué d'un seul fichier main.c et avoir en sortie un exécutable nommé *monProgramme*

gcc ***–lpthread*** *main.c –o monProgramme*

Ps. Il faut ajouter **#include <pthread.h**> au début de vos fichiers

Manipulation (1)

Création d'un Thread

Avant de créer un thread, il faut déclarer une variable le représentant par **pthread_t** (qui est, sur la plupart des systèmes, un **typedef** d'**unsigned long int**).

SYNOPSIS

```
#include <pthread.h>

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

Explication du prototype

- La fonction renvoie une valeur de type int : 0 si la création a été réussie ou une autre valeur si il y a eu une erreur (Identifiant du thread).
- Le premier argument est un **pointeur vers l'identifiant du thread** (valeur de type pthread_t). C'est un passage par adresse.
- Le second argument désigne les **attributs du thread**. Vous pouvez choisir de mettre le thread en état joignable (par défaut) ou détaché, et choisir sa politique d'ordonnancement (usuelle, temps-réel...). Dans nos exemples, on mettra généralement **NULL**.
- Le troisième argument est un pointeur vers la fonction à exécuter dans le thread. Cette dernière devra être de la forme **void *fonction (void* arg)** et contiendra le code à exécuter par le thread.
- Le quatrième argument est l'argument à passer au thread ou à la **fonction start_routine**.

RETURN VALUE

On success, **pthread_create**() returns 0; on error, it returns an error number, and the contents of **thread* are undefined.

Manipulation (2)

Suppression d'un Thread

A la fin de chaque utilisation, le thread est supprimé.

SYNOPSIS

```
#include <pthread.h>

void pthread_exit ( void *ret ) ;
```

En argument, la valeur qui doit être retournée par le thread, et doit être placée en dernière position dans la fonction du thread appelant.

Annulation d'un Thread

Annulation d'un thread à partir d'un autre.

SYNOPSIS

```
int pthread_cancel ( pthread_t thread ) ;
```

L'argument de cette fonction est le thread à annuler. Elle renvoie **0***(zéro)* si elle réussie ou la valeur **ESRCH** si aucun thread ne correspond à celui passé en argument.

Applicatif 1

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

void *thread_1(void *arg){
                printf("Nous sommes dans le thread.\n");
                pthread_exit(NULL);
                }

int main(void){
        pthread_t thread1;
        printf("Avant la création du thread.\n");
        if(pthread_create(&thread1, NULL, thread_1, NULL) == -    1) {
                perror("pthread_create");
                return EXIT_FAILURE;
                }
        printf("Après la création du thread.\n");
        return EXIT_SUCCESS;
        }
```

Affichage :

```
Avant la création du thread.
Après la création du thread.
```

- Création du thread sans l'exécuter.
- Le thread principal ne va pas attendre de lui-même que le thread se termine avant d'exécuter le reste de son code.
- Par conséquent, il va falloir lui en faire la demande via la fonction **pthread_join**

Prototype d'attente de la fin d'un thread

```
#include <pthread.h>

int pthread_join(
        pthread_t th,
        void **thread_return
        );
```

Paramètres :

- Identifiant du thread ou thread à attendre
- Un pointeur ≡ Valeur de retour de la fonction du thread **th**.
  L'appel de cette fonction met en pause l'exécution du thread appelant jusqu'au retour de la fonction. Si aucun problème n'a eu lieu, elle retourne **0** et la valeur de retour du thread est passé à l'adresse indiquée (second argument) **si elle est différente de NULL**. En cas de problème, la fonction retourne des valeurs d'erreur :

  - **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.

Applicatif 2

Programme qui crée un thread demandant un nombre à l'utilisateur, l'incrémentant une fois puis l'affichant dans le ***main***.

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

void *thread_1(void *arg){
                                printf("Nous sommes dans le thread.\n");
                                pthread_exit(NULL);
                                }

int main(void){
        pthread_t thread1;
        printf("Avant la création du thread.\n");

        if (pthread_create(&thread1, NULL, thread_1, NULL)) {
                perror("pthread_create");
                return EXIT_FAILURE;
        }

        if (pthread_join(thread1, NULL)) {
                perror("pthread_join");
                return EXIT_FAILURE;
        }

        printf("Après la création du thread.\n");

        return EXIT_SUCCESS;
    }
```

Affichage :

Avant la création du thread.
Nous sommes dans le thread.
Après la création du thread.

Applicatif 3

1- Implémenter le programme ci-dessus.

- Mettre en œuvre une boucle itérative pour vérifier l'évolution des threads. Faire une analyse.
- Introduire les mutex. Quelle remarque pouvez vous en tirer ?

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

void *print_message_function( void *ptr );

main()
{
   pthread_t thread1, thread2;
   char *message1 = "Thread 1";
   char *message2 = "Thread 2";
   int  iret1, iret2;

   iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
   iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);

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

   printf("Thread 1 returns: %d\n",iret1);
   printf("Thread 2 returns: %d\n",iret2);
   exit(0);
}

void *print_message_function( void *ptr )
{
   char *message;
   message = (char *) ptr;
   printf("%s \n", message);
}
```

Wait until thread returns

Threads terminate by:
- explicitly calling pthread_exit
- letting the function return
- a call to the function exit which will terminate the process including any threads.
- canceled by another thread via the pthread_cancel routine

2- On considère le programme suivant :

```
void * fsomme ( void * arg) ;
void * fproduit ( void * arg) ;


int main ( int argc , char * argv []) {
        int i ;
        pthread_t fils1 , fils2 ;
        char * temp ;
        temp = ( char *) malloc ( sizeof ( int ) + sizeof ( char)) ;
        sprintf ( temp ,"%d " ,10) ;

        if ( pthread_create ( &fils1 , NULL , fsomme , ( void *) temp )) {
                perror ( " pthread_create somme ") ;
                }
        if ( pthread_create ( &fils2 , NULL , fproduit , ( void *) temp )) {
                perror ( " pthread_create produit ") ;
                }

        printf ( " Sortie du main \n") ;
        pthread_exit (0) ;
}

void * fsomme ( void * arg ) {
    int i , somme=0 ;
    int n = atoi(( char *) arg) ;
    for (i=0 ;i <= n ;i ++) somme = somme + i ;
        printf ( " Somme = % d\n" , somme) ;
    pthread_exit(0) ;
}

void * fproduit ( void * arg ) {
    int i , produit=1 ;
    int n = atoi (( char *) arg) ;
    for (i=1 ;i <= n ;i ++) produit = produit * i ;
        printf ( " Produit = %d\ n" , produit) ;
    pthread_exit(0) ;
}
```

- Modifier le code pour que le *main* manipule un tableau de 2 fonctions (somme et produit) et lance les deux threads par une boucle.

3- Mettre en œuvre un programme qui utilise deux threads. Le premier thread demande un entier à l'utilisateur et renvoie la valeur en sortie. Le deuxième thread utilise cette valeur et l'incrémente 10 fois en affichant le résultat de chaque incrémentation.