	Matière : Système Exploitation 2 Travaux pratiques	Niveau : LF1-INFO
	TP 4 Les Threads	**AU.** 2024-2025

Exercice 1 :

Que fait ce programme ? Expliquez.

Pour compiler un programme utilisant des threads POSIX, vous devez utiliser l'option de compilation **-pthread**

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

void *my_thread_process (void * arg)
{
  for (int i = 0 ; i < 5 ; i++) {
  printf ("Thread %s: %d\n", (char*)arg, i);
  sleep (1);
}
  pthread_exit (0);
}

main (int ac, char **av)
{
pthread_t th1, th2;
void *ret;
   if (pthread_create (&th1, NULL, my_thread_process, "1") < 0) {
   fprintf (stderr, "pthread_create error for thread 1\n");
   exit (1);
}
   if (pthread_create (&th2, NULL, my_thread_process, "2") < 0) {
   fprintf (stderr, "pthread_create error for thread 2\n");
   exit (1);
}

(void)pthread_join (th1, &ret);
(void)pthread_join (th2, &ret);
}
```

Exercice 2 :

Que fait le programme ci-après ?

```
      #define MAX 10
void *fonction (void *arg)
{
printf ("%d\n", * (int *) arg) ;
return NULL ; }
void erreur (char *msg)
{
perror (msg) ; exit (1) ;}

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

for (i = 0 ; i < MAX ; i++)

if (pthread_create (&tid [i], NULL, &fonction, (void *) &i) == -
1)
    erreur ("pthread_create") ;

for (i = 0 ; i < MAX ; i++)
if (pthread_join (tid [i], NULL) == -1)

    erreur ("pthread_join") ;}
```

Exercice 3 :

Ecrire un programme en c ayant le comportement suivant :

1. Des threads sont créées (leur nombre étant passé en paramètre lors du lancement du programme) ;
2. Chaque thread affiche un message (FREE PALESTINE !) ;
3. Le thread principal attend la terminaison des différents threads créées.

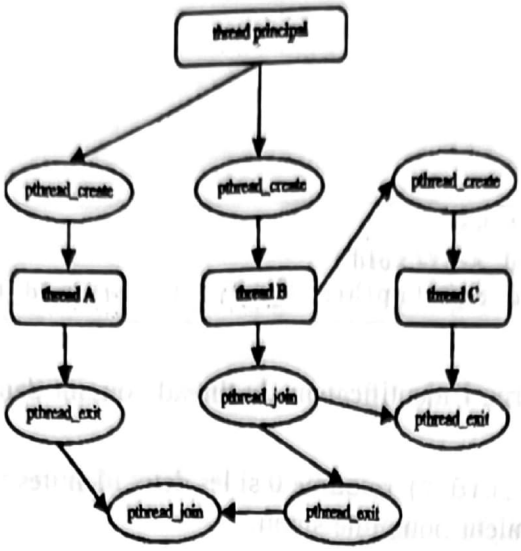
Exercice 4 :

Ecrire un programme qui crée deux threads pour exécuter chacun une fonction. Le premier thread affiche des étoiles (*) et le deuxième affiche des dièses (#).

Exercice 5 :

La figure suivante illustre un exemple de création de threads. Le thread principal crée deux threads A et B et attend la fin de leurs exécutions. Un de ces threads crée lui-même un thread C qu'il attendra avant de terminer.

Écrire un programme qui réalise l'exemple suivant.



Exercice 6 :

On aimerait implémenter un programme créant un thread pour calculer la somme d'un tableau d'entiers. Ce thread doit calculer la somme des entiers du tableau et le programme principal (main) s'occupera d'afficher le résultat.

Les points suivants sont à respecter :

- Aucun argument n'est passé au thread.
- Le thread ne retourne aucune valeur.
- Les entiers sont entrés sur la ligne de commande (chacun séparé par un espace).
- La somme est stockée dans une variable globale.
- N'oubliez pas de vérifier qu'aucune des fonctions utilisées n'échoue ; en cas d'erreur, veuillez afficher le résultat sur le canal d'erreur (stderr).

Annexe :

Création d'un thread:

```
#include <pthread.h>
int pthread_create(
    pthread_t * p_tid,              /* Pointeur sur identite du thread */
    pthread_attr_t * p_attr,        /* NULL : attributs par defaut */
    void * (*fonction)(void * arg), /* Fonction executee par le thread */
    void * arg                      /* Parametre de << fonction >> */
);
```

Identité d'un thread:

Chaque processus a un numéro unique, le pid, qui est renvoyé par la primitive getpid(). Tout processus est lui-même décomposé en threads qui ont chacun leur identifiant unique pour

un même processus, le tid.

```
#include <pthread.h>
pthread_t pthread_self(void);
int       pthread_equal(pthread_t tid_1, pthread_t tid_2);
```

1. `pthread_self()` retourne l'identificateur du thread courant dans le processus courant (le `tid`).
2. `pthread_equal(tid_1,tid_2)` retourne 0 si les deux identités transmises en argument sont identiques et une valeur non nulle sinon.

Terminaison d'un thread:

```
#include <pthread.h>
void pthread_exit(void * p_status);
int  pthread_detach(pthread_t tid);
```

1. `pthread_exit(p_status)` termine l'activité qui l'a appelée en indiquant une valeur de retour à l'adresse `p_status`. Cette primitive ne peut jamais retourner dans le thread qui l'a appelée.
2. `pthread_detach(tid)` indique au système qu'il pourra récupérer les ressources allouées au thread d'identifiant `tid` lorsqu'il terminera ou qu'il peut récupérer les ressources s'il est déjà terminé. Cette primitive ne provoque pas la terminaison du thread appelant. Elle retourne 0 en cas de succès, un code d'erreur sinon.

Synchronisation d'un thread:

Les threads disposent dans les grandes lignes des mêmes outils de synchronisation que les processus. Le premier d'entre eux est l'attente passive de la fin d'une autre activité qui rappelle les primitives `wait()` ou `waitpid()` liées aux processus. La primitive permettant ce comportement est la suivante :

```
#include <pthread.h>
int pthread_join(pthread_t tid, void ** status);
```