ASR TD4
Threads

1 Lancement de threads

Le programme suivant délègue le lancement de la fonction `fct` à un thread :

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

void * fct()
{
  puts( "Hello thread" );
}

int main()
{
  pthread_t thread;
  pthread_create( &thread, NULL, fct, NULL );
  puts( "Hello main" );
  pthread_join( thread, NULL );
  return 0;
}
```

1. Dans quel ordre vont s'afficher les 2 messages ?
2. Que se passe t'il si on ajoute un appel à la fonction `sleep` (`man 3 sleep`) avant l'affichage dans la fonction `main` ?
3. Que se passe t'il si on commente l'appel à `pthread_join` ?

1. Ajouter d'autres threads effectuant des affichages différents à ce programme.
2. Dans quel ordre sont affichés ces messages ? Cet ordre peut-il changer entre plusieurs exécutions ?
3. Créer 100 threads dans votre programme affectuant chacun une attente d'une seconde. Combien de temps va mettre l'exécution de ce programme ?

2 Observation des threads

Créer un programme créant 2 threads effectuant des attentes respectivement de 20 et 30 secondes à l'aide de la commande `sleep`, le processus père attendant la fin de l'exécution des threads. Utiliser la commande `htop` pour observer leur exécution.

- Passer l'affichage en mode arborescence de processus à l'aide de la touche `F5`.
- Rechercher votre processus dans la liste à l'aide de la touche `F3`.

1. Comment apparaissent les threads dans `htop` ?
2. Identifier leurs `pid`.

3 Variables globales et locales

1. Écrire un programme possédant une variable globale et un thread modifiant cette variable, puis effectuer un affichage de cette variable par le processus père avant l'appel à `pthread_join` et un autre affichage après cet appel. Les affichages sont-ils identiques ? Peuvent-ils être différents ?

2. Ajouter un thread supplémentaire effectuant un calcul différent sur cette même variable globale. Le résultat final est-il identique ? Quels sont les résultats possibles ?

4 Passage de paramètres

La fonction `pthread_create` prend en paramètre une fonction de type `void * fct( void * )` ce qui impose de ne passer qu'un paramètre à cette fonction et de le passer par son adresse. Le type de cette adresse doit également être converti (cast) en `void *` :

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

void * fct( void * args )
{
  printf( "Hello thread, arg=%i\n", *(int *)args );
}

int main()
{
  pthread_t thread;
  int arg = 123;
  pthread_create( &thread, NULL, fct, (void *)&arg );
  puts( "Hello main" );
  pthread_join( thread, NULL );
  return 0;
}
```

Dans le cas du passage de plusieurs paramètres, ceux-ci doivent être rassemblés dans une structure :

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

struct argstype { int a, b; };

void * fct( void * args )
{
  struct argstype * p_args = (struct argstype *)args;
  printf( "Hello thread, arg0=%i, arg1=%i\n", p_args->a, p_args->b );
}

int main()
{
  pthread_t thread;
  struct argstype args = { 14, 42 };
  pthread_create( &thread, NULL, fct, (void *)&args );
  puts( "Hello main" );
  pthread_join( thread, NULL );
  return 0;
}
```

On souhaite identifier les threads lancés par une valeur entière unique pour chaque thread. Donner un code permettant de créer 100 threads numérotés de 0 à 99 et vérifier qu'un même identifiant n'est pas attribué plusieurs fois.