Sujet de TP 3 Système 2 - Programmation système L3Info/UFR ST/UFC

Programmation multithreadée (C) - 1

V. Felea & A. Hugeat & E. Merlet

Les exercices sont orientés sur la manipulation de base des threads C norme POSIX (sans synchronisation). Exception faite pour le premier exercice, qui traite des pointeurs de fonction. La notion de pointeur de fonction est nécessaire pour la création des threads en C. Une présentation succincte de cette technique est donnée ci-après.

Pointeur de fonction - description de la technique

Un pointeur est une variable contenant une adresse mémoire. Les pointeurs peuvent également contenir l'adresse d'une fonction, c'est ce qui est appelé un *pointeur de fonction*. Cette dernière peut ainsi être passée en paramètre à une autre fonction et être appelée.

1. Déclaration d'un pointeur sur fonction

Syntaxe : `type_t (*pt_fct)(... type paramètres ...);`

Pointeur sur une fonction

- renvoyant un résultat de type `type_t`
- de nom `pt_fct`
- ayant des paramètres dont les types sont donnés entre les parenthèses (`void` si absence de paramètres)

Exemples :

```
int (*fct)();
void (*fct_param)(int, char);
```

2. Affectation d'un pointeur sur fonction

```
pt_fct = nom_fonction;  /* notation actuelle */
pt_fct = &nom_fonction; /* notation "cohérente" avec la notion d'adresse */
```

3. Appel (syntaxe) : `(*pt_fct)(... paramètres effectifs ...)`

Exemples :

```
(*fct)();
(*fct_param)(10,'a');
```

Exemple complet :

```
#include <stdio.h>
int somme(int i, int j) { return i+j; }

int main(int argc, char* argv[]) {
  int x, y;
  /* déclare un pointeur de fonction */
  int (*pSomme)(int, int);
  pSomme = somme; /* initialise pSomme avec l'adresse de la fonction somme */
  /* pSomme = &somme; */                  /* autre notation moins utilisée */

  printf("Entrer deux entiers : "); scanf("%d%d", &x, &y);
  printf("Leur somme : %d\n", (*pSomme)(x,y));
  return 0;
}
```

4. Fonction comme paramètre d'une autre fonction

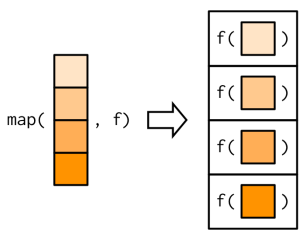
Le pointeur de fonction permet en particulier de généraliser un traitement, en donnant le traitement à réaliser (la fonction) comme paramètre d'une fonction. Un exemple est le tri des éléments d'un ensemble homogène : soit pour rendre générique le type de tri (ordre croissant ou ordre décroissant), soit pour rendre générique le type d'éléments sur lesquels le tri est appliqué.

Passage d'une fonction comme paramètre d'une autre fonction
Pour une fonction de prototype : `type fonction(type_1,...,type_n);`
le type de son pointeur de fonction est : `type (*)(type_1,...,type_n)`

Exercices

1. **Fonctions en paramètre**

   - Écrire une fonction `mult` qui multiplie par l'entier `fact`, un entier `x` et renvoie la valeur obtenue.
   - Le principe d'une fonction `map` (voir la figure) est d'appliquer un traitement donné (f dans la figure) à chaque élément d'un ensemble homogène d'éléments.



   Écrire la fonction `map` appliquant une fonction binaire sur chaque élément d'un tableau unidimensionnel (d'entiers, de taille donnée) en le modifiant.
   Les paramètres de la fonction `map` sont : la fonction binaire à appliquer, le tableau unidimensionnel et son nombre d'éléments, ainsi que le deuxième opérande de la fonction binaire (le premier opérande étant l'élément du tableau).

- Écrire le programme principal qui utilise `map` pour appliquer la fonction `mult` à un tableau unidimensionnel d'entiers.
  Exemple. Le map de la fonction `mult` sur le tableau {1,3,5} avec un paramètre `fact` à 10 modifie le tableau à {10,30,50}.

2. **Affichage concurrent**

   Q1 (fonction sans paramètre) Écrire un programme qui crée deux threads. Chaque thread affiche son identifiant toutes les secondes ainsi que le nombre de fois que le thread a écrit. Après 10 affichages chacun, les threads devraient terminer. Le programme principal doit attendre la terminaison des deux threads avant de continuer.

   Indication. La fonction `pthread_self` permet d'obtenir l'identifiant du thread courant.

   Rappel prototype `pthread_create`**.**
   ```
   int pthread_create(pthread_t* thread, const pthread_attr_t* attr, void*
   (*start_routine)(void*), void* arg);
   ```
   Il convient d'utiliser ce prototype pour la routine appelée :
   ```
   void* nom_fct(void* ptr);
   ```
   Note. À ajuster le nom de la routine du thread (*`nom_fct`*).

   L'appel de cette fonction dans un fil d'exécution indépendant est réalisé par l'intermédiaire de la fonction `pthread_create` (cas d'une fonction sans paramètres) :
   ```
   ret = pthread_create(&thread1, NULL, nom_fct, NULL);
   ```

   Q2 (fonction avec paramètre) Modifier la solution précédente pour que le nombre d'affichages maximum soit donné par le programme principal.

3. **Générateur de tableau unidimensionnel aléatoire**
   Écrire un programme qui permet d'initialiser avec des valeurs aléatoires un tableau unidimensionnel d'entiers. Le traitement sera réalisé par un thread. Chaque valeur aléatoire entière doit être comprise entre deux limites données par le programme principal, qui obtient ces valeurs depuis les arguments en ligne de commande.