Sujet du TP n° 5

Placez-vous dans le répertoire **PSE/TP5**.

Les threads (transparents 95 à 105)

Exercice 1

- copiez le fichier **exemples/slide099.c** dans le répertoire **TP5** et renommez-le **exercice1.c**
- compilez-le (tapez **make**) puis testez-le.
- que se passe-t-il ? Rien
- corrigez. Il faut rajoute l'attente de la fin du thread

Exercice 2

- écrire un programme qui crée 10 threads numérotés de 1 à 10, le rôle de chaque thread étant de calculer la somme des entiers de 1 jusqu'à son numéro; à la fin de l'exécution des threads, le thread principal récupère les résultats des calculs et les affiche (la transmission des résultats se fait pas des variables globales).

Application multithread

Copier dans TP5 l'application client/serveur réseau du **TP3** : **serveur.c** et **client.c**.

Comme nous l'avons constaté dans le TP3, les demandes de connexion des clients sont toujours acceptées par Linux mais mises en file d'attente si le serveur est occupé à dialoguer avec un client.

But : le serveur doit pouvoir dialoguer avec plusieurs clients simultanément.
Pour que les connexions des clients soient traitées sans délai par le serveur, celui-ci devra créer un nouveau thread à chaque connexion acceptée; ce thread sera chargé de traiter la session de dialogue avec le client pendant que le thread principal se remet en attente d'une nouvelle connexion.
Le code du serveur sera réorganisé selon l'algorithme suivant :

```
THREAD PRINCIPAL (main)
  ouverture journal
  initialisation réseau : socket, bind, listen

  BOUCLE infinie
    connexion d'un client (accept)
    création thread pour la session de dialogue avec le client
  FIN BOUCLE

  fermeture socket ecoute
  fermeture journal
  exit

THREAD SESSION DE DIALOGUE AVEC UN CLIENT
  TANT QUE fin non demandée
    lire ligne sur le canal
    gerer lignes particulières "fin" et "init"
    ecrire ligne dans journal
  FIN TANT QUE

  fermeture canal
  exit thread
```

Les données de chaque thread (ident, canal, ...) devront être stockées dans des variables non locales. Pour cela, vous disposez du module **datathread** qui permet de créer une liste chainée de structures, chaque structure contenant les données d'un thread. Ce module est présent dans les répertoires **modules** et **include**. Dans ce dernier :

- le fichier **datathread.h** contient les déclarations des structures et fonctions permettant la gestion de la liste chaînée
- le fichier **dataspec.h** contient la déclaration d'une sous-structure du maillon contenant les données spécifiques d'un thread (identifiant du thread, canal pour la communication, etc).

A faire

- réorganiser le serveur suivant les directives ci-dessus, le client quant à lui n'étant pas modifié
- tester pour vérifier la possibilité d'avoir plusieurs clients dialoguant simultanément avec le serveur

(c) Philippe Lalevée, 2023-2024.