PR6 – Programmation réseaux
TP n° 8 : Processus légers et verrous en C

Exercice : Un client et un serveur pour un service de tuteurs

On souhaite programmer un serveur et un client afin de gérer un ensemble de tuteurs disponibles pour aider, à la demande, des étudiants.
Le serveur devra donc maintenir à jour une liste chaînée de tuteurs disponibles, où un tuteur est représenté par son identifiant et sa discipline.

1. Le serveur attend les messages des clients sur son port. Les messages peuvent être de 5 types :

- `disp` signifie que le client demande la liste des tuteurs disponibles. Le serveur renvoie donc au client cette liste. Pour ce faire, il commence par envoyer au client un premier message contenant juste le nombre n d'éléments de la liste suivi du caractère `!` : `<n>!`. Puis, il envoie ensuite n messages du type : `<identifiant> <discipline>!`. Afin que la liste ne change pas de taille entre le moment où sa taille est envoyée au client et le moment où la liste est transmise, vous commencerez par effectuer une copie de la liste en utilisant le mécanisme de vérouillage, puis vous enverrez au client le nombre d'"éléments de la liste copiée, ainsi que la liste copiée ;
- `get <id>` signifie que le client demande l'aide du tuteur d'identifiant `id`. Le serveur lui retourne l'identifiant du tuteur si celui-ci est disponible et le retire de la liste des tuteurs disponibles. Sinon le serveur renvoie le message d'erreur `erreur` ;
- `help <disc>` signifie que le client demande un tuteur de la discipline `disc`. S'il y a un tuteur de cette discipline disponible, le serveur renvoie au client l'identifiant de ce tuteur et le retire de la liste des tuteurs disponibles. Sinon le serveur renvoie le message d'erreur `erreur` ;
- `add <id> <disc>` signifie que le client rend disponible le tuteur d'identifiant `id` rattaché à la discipline `disc`. Le serveur ajoute ce tuteur à la liste des tuteurs disponibles ;
- `quit` signifie que le client demande à se déconnecter. Le serveur ferme donc la socket de communication avec ce client.

Le serveur doit pouvoir gérer plusieurs clients en parallèle. Vous utiliserez donc des processus légers.

2. Le client se connecte au serveur et lui envoie des requêtes. Il interagit avec l'utilisateur via un terminal. Sur ce terminal, un prompt (`$`) invite l'utilisateur à entrer sa nouvelle requête tant que celui-ci ne demande pas à se déconnecter et le client y affiche les réponses. Si la requête est :

- `disp` le client affiche l'identifiant et la discipline de chaque tuteur disponible ;
- `get <id>` ou `help <disc>` alors le client affiche l'identifiant du tuteur affecté s'il y en a un. Sinon il affiche un message d'erreur ;
- `add <id> <disc>` alors le client affiche le message `Merci!` ;
- `quit` alors le client ferme sa socket et termine.

Attention, il faudra bien veiller à ce que la liste des tuteurs disponibles soit la même pour tous les clients.
Par ailleurs, deux clients peuvent demander simultanément de modifier la liste des tuteurs disponibles. Pensez à utiliser le mécanisme de verrouillage (mutex), pour éviter les réponses incohérentes du serveur.