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. Un «message» est un chaine de caractères terminé par la séquence de saut de ligne `CR` et `LF` (`\r` et `\n`).

Serveur Le serveur attend les messages des clients sur son port. Il maintient à jour une liste chaînée de tuteurs disponibles, où un tuteur est représenté par son identifiant et sa discipline. Les messages peuvent être de 5 types :

- `LIST` 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 (en ASCII) d'éléments de la liste. Il envoie ensuite n messages du type : `<id> <subj>` (identifiant et discipline).
- `ACQUIRE_ID <id>` signifie que le client demande l'aide du tuteur d'identifiant `id`. Si celui-ci est disponible le serveur lui retourne l'identifiant `id` du tuteur et le retire de la liste des tuteurs disponibles. Sinon le serveur renvoie le message d'erreur `erreur`.
- `ACQUIRE_SUBJ <subj>` 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`.
- `RELEASE <id> <subj>` 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.

Client 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 :

- `LIST`, alors le client affiche l'identifiant et la discipline de chaque tuteur disponible ;
- `ACQUIRE_ID <id>` ou `ACQUIRE_SUBJ <subj>`, alors s'il y en a un tuteur affecté le client affiche l'identifiant ; sinon il affiche un message d'erreur ;
- `RELEASE <id> <subj>`, alors le client affiche le message `Merci!` ;
- `QUIT`, alors le client ferme sa *socket* et termine.

1. Programmez le client et un premier prototype mono-filaire du serveur : présumez que au plus un client à la fois est connecté.

2. Modifiez le serveur pour gérer plusieurs clients en parallèle en utilisent des processus légers. Garantissez la cohérence en vous servent du mécanisme de verrouillage (*mutex*). Par exemple, deux clients peuvent demander simultanément de modifier la liste des tuteurs disponibles. Par ailleurs, la liste peut être mise à jour entre le moment où sa taille est envoyée au client et le moment où elle est transmise. Pour cela, vous commencerez par effectuer une copie de la liste (la section critique du code), puis vous enverrez au client le nombre d'éléments de la liste copiée, ainsi que la liste copiée.

(3) Pouvez-vous permettre accès concurrente à la liste en restreignant le verrouillage à un nombre modeste d'éléments ?