Communications Unix : TP1

Préambule :

L'objectif de ce TP est de mettre en place un modèle client/serveur à l'aide des API socket.

Les sources seront relevées à l'issue de la troisième séance de TP, et constitue votre compte rendu (vous pourrez aussi joindre des traces d'éxecution éventuellement).
Il vous est demandé de suivre d'aussi près que possible les directives du sujet (votre cahier des charges). Pensez aussi à commenter votre code et à écrire aussi proprement (réutilisablement) que possible.
On pourra limiter l'interface utilisateur à une ligne de commande, n'y consacrez cependant pas trop de temps, l'accent est mis ici sur le développement d'une librairie permettant l'interaction à travers les sockets. **L'interface utilisateur de votre programme est donc son API** (qu'il vous est demandé de soigner et de commenter). Vos executables seront construits comme un petit « main » qui se contente d'appeler les fonctions de votre librairie.

Exercice 1:

Nous allons implémenter un protocole client serveur, permettant d'accéder à des fichiers placés coté Serveur. Ce protocole permet les trames de requêtes suivantes :

101 PASSWD string
102 ACCES filename
103 QUIT

Les réponses sont composées des trames suivantes :

901 OK texte
902 INCONNU
903 NOPERM
904 NONTROUVE
905 EOF

Le protocole se déroule de la manière suivante :

- A l'initialisation de la connexion le serveur envoie une trame OK
- Le client doit avant tout s'authentifier à l'aide d'une trame PASSWD avec son mot de passe : si ce dernier est correct on renvoie une trame OK, sinon une trame INCONNU
- Toute requête du client s'il ne s'est pas authentifié donne une trame NOPERM
- Le client peut demander à accéder un fichier : trame ACCES avec le nom du fichier
    - Le serveur cherche le fichier dans le répertoire local :
        - s'il n'existe pas, on renvoie NONTROUVE
        - s'il existe on renvoie OK, suivi des lignes du fichier, suivi de EOF, le client les affiche simplement
- Le client peut terminer la session à l'aide de QUIT (on lui réponds OK avant de fermer la connexion)

Yann.Thierry-Mieg@lip6.fr

Indications complémentaires :

- Utilisez des sockets TCP (cf. support)
- Dans un fichier protocole.h, créez deux listes de *requetes* et *reponses* contenant des #define OK 901 …etc associant les codes de message à leur signification. Pour émettre des trames ou reconnaître une trame reçue utilisez exclusivement ces macros afin d'assurer la cohérence du client et du serveur.
- Chaque trame doit débuter par un int (son code) suivi d'un couple (longueur,chaine) si la trame a un argument. (PASSWD, ACCES, OK)
- Utilisez un port libre > 1024
- Utilisez les routines standard open, read, write, fread, fwrite, close, …etc. Pour accéder aux fichiers du répertoire local.
- Votre API proposera coté client :
  - Une fonction **: int getClientSocket (char * IP, int port)**
    - Se connecte a l'adresse « **IP** » sous la forme « 121.32.22.44 » sur le port « **port** » et rend le file descriptor de la socket ou -1 en erreur
  - Différentes fonctions **authentify(char* passwd,int sock), getFile(char *path,int sock), closeConnection(int sock)**
    - Elles décomposent l'action et travaillent sur « sock », supposée ouverte. Essayez de rattraper au maximum les erreurs éventuelles.
- Coté serveur :
  - Une fonction : int **getServerSocket (int port)**
    - Crée une socket d'attente de connection sur le port « port » et la place dans l'état *listen*. Rend le file descriptor de la socket.
  - Une fonction : **int accept (int sock)**
    - Accepte une connexion et rend le file descriptor associé à la socket de communication (+ récupération d'erreurs)
  - Une fonction **traitement (int sock)** éventuellement découpée en sous-fonctions qui gère la vie d'une connexion de client et suit le protocole défini plus haut.

Exercice 2

Votre serveur n'est actuellement capable de traiter qu'un seul client à la fois. Nous allons en faire un serveur multi-thread. A l'aide de *fork()* vu dans les supports ou du modèle suivant, implémentez le multi-thread coté serveur.

Comment feriez vous pour définir une fonction **newMTServer** prenant en argument un numéro de port **p** et un pointeur sur une fonction de traitement **foo** et crée un serveur multi-threadé qui attend sur le port **p** et invoque « **foo** » dans un processus séparé à chaque connection de client ? Quelle serait l'approche appropriée en programmation objet ? (répondre en commentaire dans le source)

Pour tester la cohérence de votre application et sa conformité au protocole, connectez vous avec votre client au serveur de votre voisin.