TP 10
Sockets, et logiciels client-serveur

Durée prévue : 1 séance (1h30)

L'archive fournie pour le TP contient un répertoire `echoserver1` avec : les sources de la bibliothèque `socklib`, `socklib.h` et `socklib.cpp` ; les sources `echoserver.cpp`, `echoclient.cpp`, `recv_line.h` et `recv_line.cpp` dans lesquelles vous allez travailler.

Dans le premier exercice de ce TP, nous allons utiliser le client `telnet`, qui permet de se connecter à un serveur en mode texte. Il semble que la commande soit souvent installée par défaut dans les distributions Linux (sous Debian/Ubuntu/Mint, le package s'appelle tout simplement `telnet`).

EXERCICE 1 ▶ Un serveur qui répète tout

Le but est d'écrire un serveur `echoserver` qui se met en attente d'une connexion en mode texte. Lorsqu'un client établit une connexion, le serveur doit effectuer le traitement suivant :

- le serveur doit commencer par répondre « `Bonjour !` » au client ;
- puis se mettre à lire *ligne-par-ligne* le texte qu'il reçoit de la part du client ;
- ensuite, il doit afficher chaque ligne lue sur sa sortie standard, entre guillemets droits ;
- si la ligne lue est `quit`, alors il dit "au revoir" au client, ferme la connexion, et se termine ;
- sinon, il envoie au client « `Vous avez envoyé "..."` » ;
- puis le serveur doit se remettre en attente de lecture de la ligne suivante.

Évidemment, il y a un peu de travail avant d'en arriver là ! **Encore une fois, on écrit le serveur.**

Pour cela, on doit écrire dans le source `recv_line.cpp` une fonction permettant de lire sur une socket jusqu'à ce qu'un *délimiteur* soit rencontré : les lignes lues sont séparées par ce caractère ; ce sera par défaut le retour à la ligne `'\n'`, mais cela pourrait être un autre caractère. Le prototype de la fonction sera (voir dans `recv_line.h`) :

```
int recv_line(int sd, std::string &line, char c = '\n');
```

On suppose que lors de l'appel à cette fonction, `sd` est une socket de dialogue, avec un hôte à l'autre bout qui envoie des caractères ; `c` désigne le caractère de fin de chaîne, qui sera stocké dans la chaîne lue dans `line`. La fonction retourne le nombre de caractères lus en cas de succès, 0 si la socket a déjà été fermée par le client lorsque la fonction est appelée, -1 en cas d'échec.

1) Écrivez un `Makefile` *tout simple* pour compiler l'exécutable `echoserver` à partir de `echoserver.cpp` (qui contient la fonction `main()`), de `socklib.cpp` (qui contient les définitions des fonctions de la bibliothèque) et de `recv_line.cpp`. Votre `Makefile` permettra aussi de faire le ménage (cible `clean`). Ne vous occupez pas des *warnings* à la compilation pour l'instant.

2) Une fois le serveur compilé, vous allez le lancer avec `./echoserver 8085` (que signifie le 8085 ?). Ensuite, tentez de vous connecter dessus à l'aide du client `telnet` depuis un autre terminal ; pour cela la commande est `telnet localhost 8085`. Logiquement, le serveur doit simplement vous répondre « `Bonjour !` » puis fermer la connexion, ce qui donne un affichage comme celui ci-dessous côté client. Expliquez le code qui provoque ce comportement côté serveur.

```
nlouvet@nlbook:~/lifasr5/echoserver$ telnet localhost 8085
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Bonjour !
Connection closed by foreign host.
nlouvet@nlbook:~/lifasr5/echoserver$
```

3) Dans `recv_line.cpp`, complétez la fonction `recv_line()`, en lisant simplement un par un les caractères sur la socket `sd` avec `recv()`. Prenez bien en compte les cas d'erreurs (`man recv`). Pour ajouter le caractère `t` à la fin de la chaîne `line` de type `string`, utilisez `line.push_back(t)`. **On commencera par écrire sur papier un pseudo-code…**

4) Pour tester votre fonction `recv_line()`, modifiez la fonction `main()` du serveur (au niveau du commentaire `A TERMINER`) de façon à ce qu'il attende la réception d'une ligne envoyée par le client et l'affiche sur sa sortie standard avant de fermer la connexion. Lancez votre serveur dans un terminal, puis dans un autre terminal connectez-vous au serveur avec `telnet`. Voici un exemple d'affichages produits par le serveur et par le client.

- Ce qui se passe dans le terminal du **serveur** :

```
nlouvet@nlbook:~/lifasr5/echoserver$ ./server 8085
On utilise le port 8085
create_server_socket: tentative de création d'une socket serveur sur le port 8085
create_server_socket: la socket a été créée
create_server_socket: la socket 3 est maintenant en attente à l'adresse 0.0.0.0 sur le port 8085
```

```
accept_connection: connexion depuis localhost, depuis le port 44054
Une connexion vient d'être acceptée.
J'ai reçu : Toto fait du vélo :)

nlouvet@nlbook:~/lifasr5/echoserver$
```

— Ce qui se passe dans le terminal du **client** :

```
nlouvet@nlbook:~/lifasr5/echoserver$ telnet localhost 8085
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Bonjour !
Toto fait du vélo :)
Connection closed by foreign host.
nlouvet@nlbook:~/lifasr5/echoserver$
```

5) Complétez le serveur (dans la fonction `main()`) pour qu'il effectue le traitement attendu du client (qui a été détaillé au début de l'exercice). Pour la lecture d'une ligne de texte sur la socket, utilisez la fonction `recv_line()` que vous venez d'écrire. Pour l'envoi d'une chaîne de caractères au client, utilisez la fonction suivante, qui vous a été fournie gracieusement avec la `socklib`.

```
int send_str(int sd, const std::string &str);
```

Vous consulterez les commentaires concernant cette fonction dans `socklib.h` de façon à bien gérer le cas d'erreur. Attention à bien prendre votre temps, de compiler et tester votre serveur avec `telnet` au fur et à mesure.

6) Logiquement, à chaque fois que vous tapez la touche Entrée côté client, la ligne que vous venez de taper doit être affichée par votre serveur, mais vous avez des problèmes de retour à la ligne à l'affichage (si vous avez d'autres problèmes, déboguez!)… Pour régler le problème des retours à la ligne intempestifs, il faut supprimer le retour à la ligne qui se trouve à la fin de la chaîne reçue par le serveur (notez qu'il se peut aussi la chaîne soit vide, ou que la connexion ait été coupée avant que le serveur ait reçu `'\n'`). En plus, certains clients (comme `telnet`) n'envoient pas simplement `'\n'` en fin de ligne, mais les deux octets CRLF (`"\r\n"`). Supprimez donc tous ces caractères éventuels en fin de chaîne; cela se fait bien en utilisant les méthodes `empty()`, `back()` et `pop_back()` de la classe `string`.

EXERCICE 2 ▶ Un client pour notre serveur

Dans cet exercice, vous devez réaliser l'écriture d'un client pour pouvoir interagir avec le serveur de l'exercice précédent. Ce client prendra donc la place de `telnet` que nous avons utilisé jusqu'ici.

1) Le code à compléter du client est dans le le fichier `echoclient.cpp`; commencez déjà par compléter votre `Makefile` de façon à ce qu'il fabrique aussi l'exécutable `echoclient` à partir de ce code source. N'oubliez pas que ce code fait (ou fera) intervenir des fonctions de la `socklib`.

2) Au niveau du `TODO1` dans le code fourni, modifiez le code afin que le client tente de se connecter à l'adresse qui doit être passée comme premier argument du programme `echoclient`, et sur le port passé comme deuxième argument du programme. En cas de succès, `sd` doit être une socket utilisable pour dialoguer avec le serveur; en cas d'échec, afficher un message d'erreur et terminer le programme. Vous utiliserez pour cela la fonction `create_client_socket()` au sujet de laquelle `socklib.h` contient des explications.

3) Après la connexion au serveur, le client entre dans une boucle *a priori* infinie dans laquelle, à chaque itération, il va attendre une ligne de la part du serveur, puis lire une ligne au clavier, puis l'envoyer au serveur. Le client commence donc par attendre la réception d'une ligne de texte : pourquoi?

4) Au niveau du `TODO2`, modifiez le code de façon à ce que le client attende la réception d'une ligne de texte terminée par `'\n'` de la part du serveur, puis l'affiche sur sa sortie standard.

5) Le client lit une ligne entrée par l'utilisateur sur son entrée standard grâce à `getline(cin, line)`. Au niveau du `TODO3`, modifiez le programme de façon à ce que, si l'utilisateur entre la commande `quit`, alors le client envoie cette commande au serveur puis sorte de la boucle `while(1)` pour se terminer. Attention, la chaîne obtenue grâce à `getline()` ne se termine pas par un `'\n'`, alors que le serveur attend précisément un `'\n'` pour marquer la fin d'un message.

6) Il ne reste plus qu'à modifiez le code au niveau du `TODO4` pour que, s'il n'est pas sorti de la boucle, le client envoie la ligne `line` au serveur. Attention, tout comme le client, le serveur attend toujours une ligne qui se termine par un `'\n'`!

7) Compilez votre programme, et testez-le. Pour cela, lancez comme dans l'exercice précédent le serveur dans un terminal avec `echoserver 8085`, puis dans un autre terminal lancez votre client avec `echoclient localhost 8085`. Si vous n'obtenez pas le même comportement qu'avec `telnet` dans l'exercice précédent, il faut déboguer.