TP - ASR5, Système d'Exploitation

Les sockets

Mars 2018

I Prise en main

Sur la page du cours vous pouvez télécharger les fichiers `client.cpp` et `serveur.cpp`. Ce sont des programmes dans lesquels le client envoie au serveur tout ce qui est tapé au clavier, ce qu'ensuite le serveur affiche. Vous pouvez observer les sockets utilisées par ces programmes grâce à la commande `netstat` dont une documentation est à cette adresse : `http://perso.univ-lyon1.fr/fabien.rico/site/systeme:commande#reseau`

Q.I.1) - Compilez les 2 programmes et testez-les.

1(a) - Quelle(s) adresse(s) et quel(s) port(s) sont utilisés par le client et le serveur ? Où cela est-il modifiable dans le code.

1(b) - Peut-on lancer plusieurs serveurs ? Pourquoi ? Quelle erreur cela génère-t-il ?

1(c) - Peut-on lancer plusieurs clients ? Quand un client est connecté, peut-il transmettre des messages ?

1(d) - Pouvez-vous utiliser d'autres clients avec ce serveur ? Par exemple la commande `telnet` ou la commande `nc`.

1(e) - Si vous utilisez une machine de l'université et pas un ordinateur connecté au wifi, vous pouvez tester la même chose entre 2 machines différentes. [1]

II Simplification du code

Les codes de création du client et du serveur sont assez illisibles. Pour simplifier vos travaux futurs, nous vous proposons une bibliothèque `socklib` qui rassemble le code de créations de connexions en 3 fonctions (dans l'espace de nom socklib) :

- int CreeSocketServeur(const std::string &port); qui crée une socket d'écoute de serveur utilisant le port `port` et toutes les adresses disponibles de la machine. Cette fonction renvoie la socket ou -1 en cas d'erreur.
- int AcceptConnexion(int s); accepte un client qui tente de se connecter sur la socket d'écoute `s`. Cette socket retourne la socket de dialogue avec le serveur ou -1 en cas d'erreur.
- int CreeSocketClient(const std::string &serveur, const std::string &port); crée une socket pour un client et tente la connexion sur `serveur:port`. Cette fonction retourne la socket de dialogue avec le serveur ou -1 en cas d'erreur.

Ces 3 fonctions sont simplement une copie des codes précédents permettant la mise en place de la connexion.

À partir de ces 3 fonctions et du code précédent, vous devez créer un programme similaire à `nc`, c'est-à-dire un programme qui peut être à la fois serveur ou client. La différence dépendra de la façon dont vous allez le lancer. Pour cela vous allez utiliser les arguments de la fonction `main()` : `argc` et `argv`. Vous pouvez voir à cette adresse une explication détaillée de la méthode d'utilisation : `https://openclassrooms.com/courses/les-parametres-de-la-fonction-main`

1. Attention, les configurations classique du wifi filtrent certains ports et interdisent la communication entre machines différentes connectées au réseau. En général, les port web (80, 443), DNS ... sont autorisés mais il faut être administrateur au moment du lancement du programme pour pouvoir se placer en écoute sur ces ports, car ils sont réservés.

Q.II.1) - Faites un programme qui lit les arguments de la ligne de commande et :
- lorsqu'il est appelé avec 1 argument, ce programme doit afficher qu'il est serveur. Son argument est alors le numéro ou le nom du port ;
- lorsqu'il est appelé avec 2 arguments, ce programme doit afficher qu'il est client. Ses argument sont le serveur à contacter et son port ;
- affiche une erreur dans tous les autres cas.

Par exemple :

```
# en tant que serveur
> ./discussion.exx 8080
Je suis serveur sur le port 8080
# en tant que client
> ./discussion.exx localhost 8080
Je suis client je contacte localhost:8080
# en oubliant les arguments
> ./discussion.exx
usage ./discussion.exx <port> : pour le serveur
      ./discussion.exx <serveur> <port> : pour le client
```

Ensuite grâce aux fonctions de `socklib.hpp` et `socklib.cpp`,

Q.II.2) - modifiez le serveur pour attendre un client sur le port demandé ;

Q.II.3) - modifiez le client pour se connecter au serveur ;

Q.II.4) - ajoutez une boucle dans laquelle le client lit les lignes tapées par l'utilisateur et les envoie sur la socket (fonction `send()` ou `write()`) ;

Q.II.5) - ajoutez une boucle dans laquelle le serveur lit les données envoyées par le client et les affiche.

Ce code sera utilisé au prochain TP. **Faites attention a bien faire une copie de ce fichier si vous poursuivez le TP**.

III Dialogue en réseau (si vous êtes en avance)

Pour le moment, le dialogue n'est possible que dans un sens. Vous devez modifier le code pour permettre un dialogue dans les 2 sens. C'est-à-dire que le client et le serveur doivent à la fois lire les lignes tapées et les envoyer sur la socket ainsi que lire les données de la socket et les afficher.

Q.III.1) - Cela pose-t-il un problème ? Lequel ?

Q.III.2) - Quelle solution proposez-vous ?

Q.III.3) - Implémentez la solution.