TP - ASR5 système d'exploitation

Les sockets

28 fev 2016

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, alors que le serveur l'affiche. Vous pouvez observer les sockets utilisées par ces programmes, grâce aux commandes :

- netstat −tlnp qui affiche tous les serveurs en écoute sur votre machine.
- netstat −tnp qui affiche les connexions TCP actuellement en cours.

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.

- Le serveur utilise `8080` comme port d'écoute et par défaut toutes les adresses IPv4 de la machine sont utilisées.
- Le client se connecte sur l'adresse `localhost` et le port `8080`. Il essaye d'abord de se connecter en ipv6 (`::1`, mais le serveur ne répond pas). Il essaye ensuite en IPV4 `127.0.0.1` et le serveur répond. Le port du client est tiré au hasard, l'adresse IP utilisée par ce client est celle du protocole IP testé (127.0.0.1 pour le protocole ipv4).

Tous ces paramêtres sont modifiables au niveau de la fonction getaddrinfo. Dans le code proposé il s'agit de constantes définies par des macro *#define*.

1(b) - 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

Le code de création du client et du serveur est assez illisible. Pour simplifier vos travaux futurs, nous vous proposont une librairie `socklib` qui rassemble le code de créations de connexions en 3 fonctions :

- 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 focntion 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 renvoie 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 connection sur `serveur:port`. Cette fonction renvoie la socket de dialogue avec le serveur ou -1 en cas d'erreur.

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

À partir de ces 3 fonctions et du code précédant, vous devez créer un programme qui sera à la fois serveur et client. La différence dépendra de la façon dont vous allez le lancer. Pour cela vous

1. Attention, le wifi étudiant interdit la connexion entre machines et sur des ports différents de 80 ou 443. Pour faire le test avec vos PCs, vous pouvez utiliser ces ports, mais il faut être administrateur au moment du lancement du programme.

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`

Q.II.1) - Faites un programme qui :
- informe qu'il est serveur et le port utilisé lorsqu'il est appelé avec 1 argument ;
- informe qu'il est client et le serveur à contacter lorsqu'il est appelé avec 2 arguments ;
- signale une erreur dans tous les autres cas.

Ensuite grâce aux fonctions de `socklib.h`,

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 affiches.

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

III Dialogue en réseau

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 ?

2 choses peuvent venir à l'esprit :
- La socket est utilisées dans les 2 sens, est-ce possible ? la réponse et oui, une socket comporte 2 canaux.
- Comment faire pour que qu'un même processus lise à la fois sur l'entrée standart et sur la socket ? Les 2 opérations sont blocantes et on ne sais pas dans quel ordre il faut les faire.

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

Il y a plusieurs réponses au second problème :
- Utiliser `select` ce qui est la meilleur réponse dans ce cas car cette fonction permet d'attendre l'arrivée de données sur plusieurs descripteurs de fichier. Cela simplifierait tout ... à par la lecture.
- Utiliser un thread mais vous ne ferez ça que l'année prochaine.
- Utiliser 2 processus, un pour chaque rôle.

Q.III.3) - Implémentez la solution.