TP de programmation réseau

Olivier AUBERT

2003

1 Introduction

Les fichiers d'exemple sont disponibles pour ce TP, à l'adresse `http://www710.univ-lyon1.fr/~oaubert/tp/`. Commencez par aller récupérer le fichier `socket.tgz` contenant les exemples et extrayez les fichiers dans un répertoire approprié.

Où trouver les informations nécessaires pour réaliser ce TP ?

Manuel Les pages de manuel (standard sous UNIX) décrivent la syntaxe des fonctions utilisées. Pour avoir la syntaxe de la commande `socket`, il faut utiliser `man 2 socket`. Pour avoir des informations plus précises sur les paramètres lors d'une connexion IP, il faut utiliser `man 7 ip`.

Les pages de manuel décrivent notamment : les fichiers d'entête (`.h`) à inclure pour utiliser la fonction, la syntaxe des arguments de la fonction et la valeur de retour de la fonction (à tester impérativement pour détecter les erreurs).

FAQ Pour des problèmes particuliers sur la programmation des sockets, la réponse se trouve probablement dans la FAQ à l'adresse `http://www.developerweb.net/sock-faq/` (voir plus particulièrement les questions triées par catégorie à la fin du document).

2 Côté client

2.1 Écriture d'un client TCP simple

On veut écrire un client simple qui va se contenter de lire une ligne d'information auprès d'un serveur. Par exemple, un service UNIX standard est le service *daytime* qui retourne la date courante de l'ordinateur interrogé.

Une manière simple d'interroger ce système est via la commande `telnet` sur le port adéquat.

> **Q2.1** *Consultez le manuel de telnet pour trouver comment interroger un service particulier. Vous pouvez interroger un serveur quelconque, ou bien la machine sur laquelle vous êtes connecté en l'appelant* `localhost`.

Nous allons écrire un client simple en C qui va effectuer la même action. Le squelette du code se trouve dans le fichier `client-daytime.c`.

Il faut dans un premier temps déterminer le type de protocole que l'on peut utiliser pour interroger le service. Pour cela, consulter le fichier `/etc/services`.

> **Q2.2** *Quel est le type de protocole utilisé pour le service* daytime *? Quel est le numéro de port associé ?*

> **Q2.3** *Où peuvent se poser des problèmes de format des données numériques (big endian/little endian) ? Pourquoi n'a-t-on pas utilisé ici de fonctions de conversion telles que* `htons` *?*

> **Q2.4** *Complétez le code du client aux emplacements indiqués par des commentaires commençant par* `/* TP :`.

2.2 Écriture d'un client plus élaboré

On va faire évoluer le client précédent vers un autre type de service, requérant une interaction. Le service *finger* renvoie des informations sur l'utilisateur d'une machine.

Q2.5 *Utilisez `telnet` pour interroger le service `finger`, et comprendre comment il fonctionne (la page de manuel de `finger` peut aider).*

Q2.6 *Extrayez du programme précédent une fonction simple permettant d'ouvrir une socket sur la machine et le port précisés. Le squelette de cette fonction se trouve dans le fichier `connexion.c`.*
L'interface de la fonction sera
`int connexion-tcp (char* serveur, int port);`
Le port sera transmis dans le codage standard du client, il ne faut pas oublier de le convertir dans le codage réseau à l'intérieur de la fonction.
La valeur de retour sera le descripteur de socket obtenu, -1 en cas d'erreur.

Q2.7 *En utilisant la fonction ainsi définie, envoyez le nom d'utilisateur donné en paramètre au service* finger *et affichez le résultat de le requête.*

3 Côté serveur

Nous allons maintenant passer à l'écriture de serveurs, afin d'en étudier voir le principe.

3.1 Écriture d'un serveur simple

Nous avons écrit un client *daytime*, nous allons à présent en écrire un serveur. Le serveur va donc attendre une connexion, et renvoyer la date courante de la machine (fonctions `time` et `ctime`).

Q3.1 *Utilisez le squelette fourni dans `serveur-daytime.c` et ajoutez les appels nécessaires pour établir la socket, la mettre en état d'attente et traiter les connexions.*

Q3.2 *Lancez le serveur sur un port supérieur à 1024 et essayez de l'interroger avec la commande `telnet` ainsi qu'avec le client* daytime *écrit précédemment (ce qui peut nécessiter une recompilation pour spécifier le numéro de port).*

Q3.3 *Si ce n'est pas déjà fait, isolez la fonction de traitement de la connexion dans une fonction à part (`traite_requete`).*

3.2 Écriture d'un serveur plus élaboré

Afin de mettre en évidence les problèmes d'accès concurrent à un même serveur, on va créer un serveur simpliste d'addition, prenant en entrée deux nombres et renvoyant leur somme.

Q3.4 *Modifiez le code du fichier `serveur-addition.c` pour accomplir la tâche précisée.*

Q3.5 *Connectez-vous au serveur indiqué via la commande `telnet` et testez le bon fonctionnement du serveur.*

Q3.6 *Lancez depuis plusieurs terminaux plusieurs connexions simultanées vers le même serveur d'addition. Qu'observez-vous ?*

3.3 Écriture d'un serveur multi-connexions

Il existe plusieurs manières de gérer le problème des connexions concurrentes. Nous allons en voir la plus commune ici. Elle consiste à créer un nouveau processus (fonction `fork`) pour chaque nouvelle connexion. Le fils va alors traiter la requête (fonction `traite_requete`), tandis que le père va simplement fermer la socket de connexion et se remettre en situation d'attente.

Q3.7 *Modifiez le code de `serveur-addition2.c` de manière à créer un nouveau processus pour gérer chaque nouvelle connexion.*

Q3.8 *Lancez depuis plusieurs terminaux plusieurs connexions simultanées vers le nouveau serveur d'addition.*