INTRODUCTION AUX RÉSEAUX

TP 1 : programmation réseaux sous Linux en C

Premiers pas

Objectifs de la séance Le but de cette séance est de vous familiariser avec les appels systèmes Unix fournis par l'API de la couche transport pour la réalisation d'applications réseaux. Plus précisément, vous allez apprendre à :

- vous familiariser avec l'API de programmation réseau ;
- initier une connexion IP, envoyer et recevoir un message ;
- programmer un client et un serveur en différentes situations.

Tous les fichiers nécessaires à ce TP sont sur la page web du cours :
`http://www-lisic.univ-littoral.fr/~ramat/download/tp1-net.tar.gz`

1 Premier pas

Dans cette partie, nous allons nous intéresser au cas simple de la communication unidirectionnelle. Pour cela, nous disposons de deux programmes un "client" et un "serveur". Dans ce cas de figure, le serveur attend une requête de connexion, le client initie la connexion.

Téléchargez l'archive `tp1.tar.gz` depuis le site web du cours. Créez un répertoire pour les TPs du module, `reseaux` par exemple. Décompressez l'archive dedans avec la commande `tar zxvf tp1.tar.gz`. Ceci va créer un sous-répertoire `tp1` contenant plusieurs fichiers source.

Pour cette partie nous nous intéressons au programme client `hello-c.c` et au programme serveur `hello-s.c`. Lisez le code et familiarisez vous avec lui, il y a quelques commentaires pour vous aider. Répondez aux questions suivantes sur la feuille en fin de ce sujet.

1. Compilez les deux programmes `hello-c.c` et `hello-s.c`. Pour ceux qui l'auront oublié, la commande de compilation est : `gcc -g -Wall le-source.c -o le-nom-du-programme`
2. Démarrez un second terminal et exécutez chacun des programmes sur un terminal distinct. Vous devez lancer le serveur avant de lancer le client.
3. Décrivez ce que font les deux programmes. Dessinez le protocole du serveur et celui du client.
4. Les programmes acceptent des paramètres en ligne de commande. Lancez les avec l'option `-h` pour les connaître ;
5. Décrivez ce qui se passe quand le serveur et démarré sur un autre port, essayez avec `hello-s -p 123` et `hello-s -p 1234`. Donnez une explication de ce comportement.
6. A l'aide de la commande `/sbin/ifconfig` déterminez l'adresse IP de votre station de travail. Essayez de vous connecter au serveur en précisant son IP en ligne de commande au client avec `hello-c -a x.y.z.t`. Que se passe-t-il ?
7. Remplacez `127.0.0.1` à la ligne 24 de `hello-s.c` par votre adresse IP recompiler et réessayez. Que se passe-t-il ?
8. Remplacez maintenant avec `0.0.0.0` et réessayez. Que se passe-t-il ?
9. Remplissez le tableau donné sur la feuille réponses, et donnez une explication de ce comportement.
10. Demandez l'adresse IP de la station de vos voisins et demandez leur de lancer leur serveur. Faites de même pour eux. Essayer de vous connecter au serveur voisin depuis votre station. Refaites les mêmes manipulations qu'à la question 6 et remplissez le tableau donné sur la feuille réponses.

2 Programmation d'une API

Puisque les programmes que nous allons écrire utilisent des morceaux de code similaires, nous allons définir des fonctions génériques. Ces fonctions encapsuleront les différentes étapes et les appels systèmes de l'API.

En quelque sorte, vous allez écrire une couche supplémentaire au-dessus de l'API réseau.

Vous mettrez ces fonctions dans un fichier nommé `net_aux.c` est vous les déclarerez dans le fichier entête `net_aux.h`. Ce dernier devra être inclus par tous vos programmes.

Programmation

Voici les fonctions à écrire En cas d'échec, les procédures doivent afficher un message d'erreurs en utilisant la fonction `perror`, et terminer le processus avec la fonction `exit(EXIT_FAILURE)`. De plus, ses fonctions peuvent afficher des messages d'information si la constante `DEBUG` est définie par `#define DEBUG`.

⚠ Veillez à respecter exactement les prototypes ci-dessous. La correction se fera avec ma version.

- `int create_socket(void)`
  - Fonction : créer une socket de type TCP/IP.
  - Entrées : aucun paramètre
  - Sortie : un entier, le descripteur de fichier de la socket.
  - Gestion des erreurs : afficher un message et terminer le processus.
- `void open_connection(int sock, const char *ip, int port)`
  - Fonction : demande une connexion sur la socket `sock` avec l'hôte dont l'adresse est `ip` sur le port `port`.
  - Entrées : 1) entier, DF d'un socket, 2) chaîne de caractères, l'adresse IP, 3) entier, port.
  - Sortie : aucune
  - Gestion des erreurs : afficher un message et terminer le processus.
- `void close_connection(int sock)`
  - Fonction : fermer la connexion sur la socket `sock`.
  - Entrées : un entier, le descripteur de fichier de la socket
  - Sortie : aucune.
  - Gestion des erreurs : afficher un message et terminer le processus.
- `void sock_send(int sock, const char *msg)`
  - Fonction : envoyer la chaîne de caractères `msg` dans la socket `sock`.
  - Entrées : 1) entier, le DF de la socket de communication, 2) une chaîne de caractères, le message à envoyer.
  - Sortie : aucune.
  - Gestion des erreurs : afficher un message et terminer le processus.
- `void sock_receive(int sock, char *msg, int size)`
  - Fonction : lire un message depuis `sock` de taille au plus `size`. Ce message est copier dans `msg`. Attention ne pas oublier de terminer la chaîne de caractères `msg` par un `'\0'`, le caractère `NULL`. Le tampon `msg` doit être alloué au préalable.
  - Entrées : 1) entier, le DF de la socket de communication, 2) une chaîne de caractères, le message à lire, 3) entier, la taille maximale du message.
  - Sortie : aucune.
  - Gestion des erreurs : afficher un message et terminer le processus.
- `void start_server(int sock, const char* ip, int port)`
  - Fonction : démarre un serveur associé à la socket d'écoute `sock` à l'adresse `ip` et au port `port`.
  - Entrées : 1) entier, le DF de la socket de communication, 2) une chaîne de caractères, l'IP du serveur, 3) entier, le port d'écoute.
  - Sortie : aucune.
  - Gestion des erreurs : afficher un message et terminer le processus.
- `int wait_connection(int sock)`
  - Fonction : attendre une requête de connexion sur le socket d'écoute `sock` et retourne une nouvelle socket pour échanger avec le client connecté.
  - Entrées : entier, le DF d'une socket d'écoute.
  - Sortie : entier, le DF d'une socket.
  - Gestion des erreurs : afficher un message et terminer le processus.

Validation

Les programmes `hello2-c.c` et `hello2-s.c` disponibles dans l'archive que vous avez téléchargé utilisent les fonctions ci-dessous. Ils font exactement le même travail que les deux autres programmes précédents.

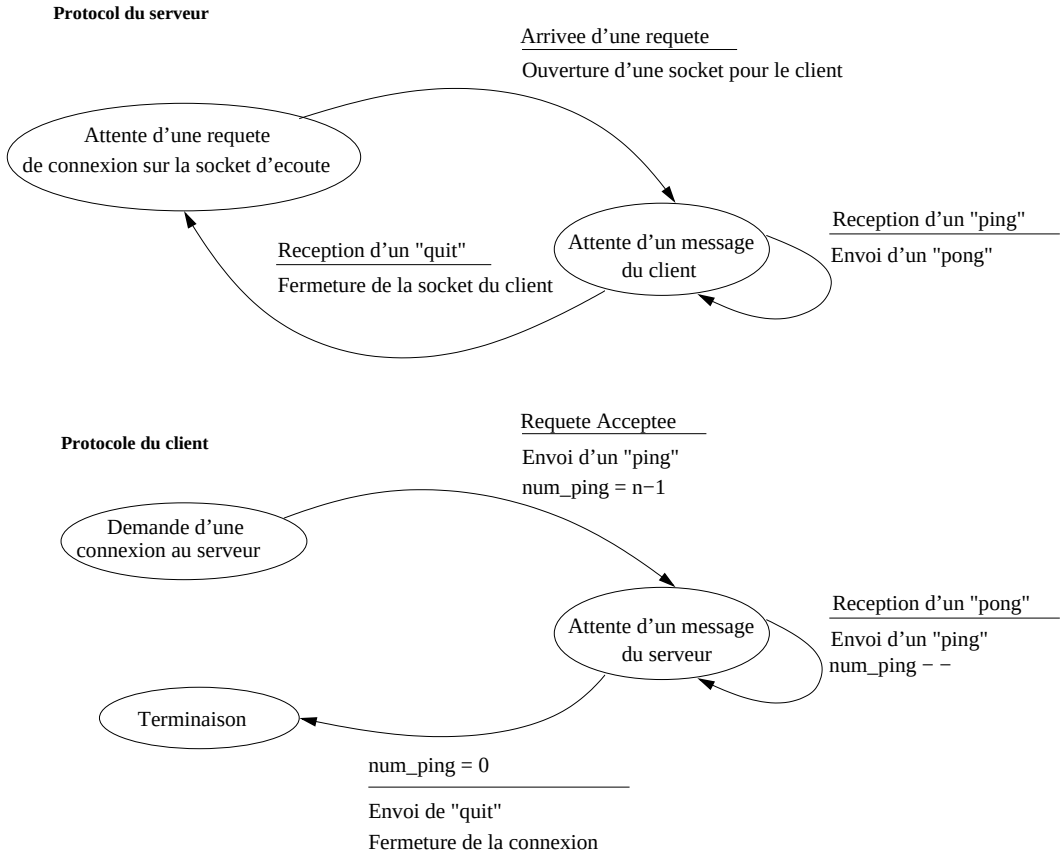
Rappel : pour compiler plusieurs code sources en un seule programme `gcc -Wall src1.c src2.c ... -o le-programme`.

⚠ Remarquez bien comment le serveur `hello2-s.c` a été modifié a afin qu'il puisse gérer plusieurs connexions successives (à ne pas confondre avec simultanées).

3 Un premier client/serveur

Dans cette partie, nous allons nous intéresser au cas où les messages passent entre le client et le serveur dans les deux directions. Pour ce faire, nous allons avoir besoin de définir un protocole de communication qui va déterminer la séquence des échanges et les comportements du serveur et du client.

Le protocole que nous allons utiliser est le suivant :



Comme vous le voyez, ce protocole est très simple, le client envoie la chaîne de caractères `ping` et le serveur envoie `pong` et ainsi de suite jusqu'à ce que le serveur reçoit un `quit` du client. A ce moment, il ferme cette connexion et se remet en écoute.

En utilisant `net_aux.c`, programmez un client et un serveur qui respectent ce protocole. Nommez vos fichier `ping-c.c` et `ping-s.c`.

⚠ 1. Le nombre de `ping` envoyés par le client sera un paramètre de la ligne de commande. Ajoutez une option `-n` comme paramètre en ligne de commande : avec `ping-c -n 10` le client envoie 10 `ping`.
2. Faites des essais avec vos voisins (attention le même protocole doit être utilisé)

Travail à fournir pour dans une semaine

Pour ce TP, les 4 fichiers suivants sont à rendre par mail (lire la procédure ci-dessous) : `net_aux.c net_aux.h ping-s.c ping-c.c`

⚠ Tous vos programmes sont à mettre dans une archive `tar` compressée par `gzip`. Vous devez nommer cette archive `tar` avec `mon_prenom.tar.gz` et l'envoyer par mail à `ramat@lisic.univ-littoral.fr`. Le sujet du mail doit commencer par `[L3-info-tp1]`, crochets inclus. Vous recevrez un accusé de réception quelques minutes après la réception. En cas d'envois multiples, la dernière version sera prise en compte.

Création de l'archive `tar` et la vérifier : Dans le répertoire `tp1` tapez la commande suivante :
`tar zcvf nom_prenom.tar.gz net_aux.c net_aux.h ping-s.c ping-c.c`

⚠ Avant d'envoyer votre travail, vérifiez le avec la commande suivante : `sh verify.sh nom_prenom tp1 1` toujours dans le répertoire `tp1`. Ce script est celui que j'utiliserai pour valider vos fichier. Il créera un répertoire `nom_prenom` dans `tp1` et en sortie vous devez avoir le résultat suivant :

```
               Name     Present Archive Compile Build   Global
-------------------------------------------------------------------
         nom_prenom     OK      OK      OK      OK      OK
```

Si au lieu des 5 `OK` il y a 1 `ER`, cela indique que votre programme a échoué à l'une des étapes. Il faudra le corriger avant de l'envoyer.

L'évaluation : Voici les points qui seront évalués :

- la compilation : sans aucun *warnings* et sans aucune erreur ;
- l'exécution : sans plantage ;
- le respect des protocoles : `ping-s.c et ping-c.c` seront compilés avec ma propre version de `net_aux.c net_aux.h`, si vos programmes respectent l'API il n'y aura aucun problème ;
- la bonne marche des programmes.

Feuille réponses

Premier pas

Question 3 :

Question 5 :

Question 6 :

Question 9 :

Dites seulement si la connexion a échouée ou réussie.

	Serveur (valeur de `ip_serveur` à la ligne 24)		
	`127.0.0.1`	l'IP de votre station	`0.0.0.0`
`hello-c -a 127.0.0.1`			
`hello-c -a l'ip de votre station`			

Votre conclusion :

Question 9 :

Dites seulement si la connexion a échouée ou réussie.

	Serveur de votre voisin (valeur de `ip_serveur` à la ligne 24)		
	`127.0.0.1`	l'IP de votre voisin	`0.0.0.0`
`hello-c -a 127.0.0.1`			
`hello-c -a l'ip de votre voisin`			

Votre conclusion :