Utilisation du réseau sous unix
Configuration de la machine, utilisation de ssh, Programmation des sockets

Objectif

L'objectif de ce TP est triple. Etudier la configuration réseau (de base) existante d'une machine sous linux. Mettre en place un outils de connexion et de transfert de fichier sécurisé (ssh) et examiner le fonctionnement de la programmation avec les sockets.

1 Etude de la configuration réseau de votre machine

On se propose d'analyser la configuration réseau de votre machine basée sur la Ethernet et les services Internet offerts.

1.1 les fichiers de configuration

- Fichier `/etc/hosts` : Editer le fichier `/etc/hosts` en utilisant la commande : `less /etc/hosts` A quoi sert ce fichier ? Pourquoi ne contient-il pas le nom de l'ensemble des machines de l'université ?
- Fichier `/etc/resolv.conf` : Editer le fichier `/etc/resolv.conf`. Qu'indique ce fichier ?
- Fichier `/etc/sysconfig/networks` : Editer le fichier `/etc/sysconfig/networks`. A quoi sert ce fichier ? A quoi correspond l'entrée loopback ?
- Fichier `/etc/services` : Editer le fichier `/etc/services`. A quoi sert ce fichier ? Retrouver le numéro de port des services ftp, telnet, ftp et SMTP.
- Fichier `/etc/protocols` : Editer le fichier `/etc/protocols`. A quoi sert ce fichier ? A quel niveau protocolaire retrouve-t-on ces valeurs ? Quel en est ainsi l'intérêt ?
- Fichier `/etc/xinetd.conf` : Editer le fichier `/etc/xinetd.conf` (ou `/etc/inetd.conf`. A quoi sert ce fichier ? Quel superdémon (superserveur) gère les services autorisés ? On pourra retrouver ce superdémon en exécutant la commande : `ps -edf | less`. Quels sont les services Internet réellement autorisés ? Analyser les fichiers de configuration sous `/etc/xinetd.d`.

1.2 les commandes liées à la configuration

- Commande `ifconfig` : En s'aidant du manuel en ligne, préciser le rôle de la commande ifconfig. Combien d'interface(s) Ethernet possède votre machine ? A quoi sert l'interface lo ? A quoi correspond le paramètre MTU ? Quelle est sa valeur ? Quelle est la valeur du masque réseau ? Quelle est la classe de réseau IP ? Quelle est l'adresse IP de broadcast ?
- Commande `netstat` : En s'aidant du manuel en ligne, préciser le rôle de la commande `netstat`.
  - A l'aide des options :`netstat -nr` préciser les passerelles liées à votre machine, quel est leur rôle ?
  - A l'aide de l'option : `netstat -a` retrouver les services UDP et TCP actifs sur le PC ainsi que la liste des connexions TCP en cours. Remarquer la notation host.numero_port et l'état courant d'une connexion TCP. (cf. Annexe).

2 Utilisation de ssh

sshd est un serveur qui permet de communiquer de façon sécurisée, en établissant un canal de communication entre lui et ses clients. Il fonctionne seul (et non comme telnet et ftp sous la dépendance de xinetd) et écoute les requêtes adressées par les clients au port 22.

Pour en savoir plus sur le serveur ssh, lire en détail `man ssh`. Les fichier de configuration sont placés dans `/etc/ssh/`.

2.1 Connexion à distance avec ssh

Syntaxe : `ssh user@serveur`, `user` étant un compte valide défini sur serveur.

Connectez-vous au compte test/test du serveur ***sis.univ-tln.fr****. A quoi correspond la question qui est posée ?*

Vous êtes dans la même situation qu'avec ***telnet****, prise de controle à distance. Vous testerez quelques commandes (listage de répertoires, ...)*

Consultez le répertoire ***.ssh*** *sur la machine cliente. Que contiennent les fichiers ? Connectez-vous à une autre machine, que constatez-vous ?*

2.2 Connexion avec une paire de clé

ssh autorise un autre mécanisme d'authentification, basé sur des clés d'authentification, une méthode cryptographique à clé publique. Chaque utilisateur désireux d'utiliser ssh avec une authentification à clé publique doit lancer la commande `ssh-keygen` (sans option) pour créer les clés d'authentification. La commande commence la création de la paire de clés (publique et privée) et demande à l'utilisateur de saisir une phrase de passe pour les protéger.

`Générez une paire de clés sur votre machine cliente à l'aide de l'utilitaire ssh-keygen -t rsa.Que constatez-localement ? Transférez la clé publique sur une autre machine de licence à l'aide de scp. Ajoutez la clé dans le fichier authorized_keys2. Essayez de vous connectez sur une autre machine (vous activerez l'option -v de ssh) ?`

2.3 Tranfert de fichiers avec scp

scp (secure copy), permet de copier des fichiers et des arborescences, en utilisant ssh pour sécuriser les transferts.

La syntaxe générale est `scp [-r] source destination`, où `source` et `destination` désigne l'ensemble des fichiers à copier ou le répertoire d'accueil. Si les fichiers sont locaux, on utilise la syntaxe habituelle, s'ils sont distants, la notation est celle de ssh : `userserveur:fichiers`.

Exemples :

- `scp -r user@serveur:fichiers rep-local` , pour copier du serveur distant les fichiers vers le répertoire rep d'accueil local
- `scp -r fichiers-locaux userserveur:rep` , pour copier les fichiers locaux vers le répertoire situé sur le serveur distant

Copier les fichiers du répertoires TEST dans votre répertoire personnel sur une des machines de licence. Copier le fichier ***titi.txt*** *à la racine du compte test de la machine* ***sis*** *en le renommant toto.txt (en une seule commande). Connectez-vous sur une autre machine de licence ou de maîtrise que constatez-vous à propos du répertoire TEST et du fichier toto.txt ? Comment ?*

2.4 Création d'un tunnel sécurisé par l'exemple

Connectez-vous en utilisant telnet sur la machine sis.univ-tln.fr. Lancez l'application xterm. Que ce passe-t-il ? Faites la même chose avec ssh.

Testez les commandes ***telnet localhost*** *et* ***telnet mail.univ-tln.fr*** *? Exécutez la commande* ***ssh -L 12345:mail.univ*** ***test@sis.univ-tln.fr*** *puis la commande* ***telnet localhost 12345****. Quel est l'intérêt de cette manipulation ? Lisez en détail l'explication des options -L et -R du man de ssh.*

Expliquer comment sécuriser un protocole avec ssh ? Donnez des exemples : smtp, pop ou imap.

3 Un premier exemple de programmation avec les sockets

3.1 Le principe

Les sockets sont basés sur une architecture client/serveur: Le serveur décide d'accepter les demandes de connection sur un port particulier, tandis que le client décide de se connecter sur le serveur sans pour autant être sûr que cela sera possible. Une fois la connection établie, les deux protagonistes peuvent communiquer à l'aide de read et write, exactement de la même manière que lorsque l'on lit/écrit dans un fichier binaire.

Un exemple est fourni : le serveur `Exemple-simple/socket_srv.c` et le client `Exemple-simple/socket_clt.c`. Vous pourrez suivre les explications en lisant les sources fournis. Pour tester l'exemple :

1. compiler socket_srv.c et socket_clt.c : `gcc socket_clt.c -o socket_clt` et `gcc socket_srv.c -o socket_srv`

2. executer le serveur sur une machine : `./socket_srv 9999`
3. executer le client sur une machine vers une autre machine: `./socket_clt 10.9.185.X 9999` avec X c ompris entre 201 et 216.

3.2 Le serveur

Avant toute chose, de la même manière que lorsque l'on ouvre un fichier, le serveur se doit d'obtenir un identificateur de socket, ce qu'il fait à l'aide de la fonction `socket()`.

`socket()`

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int family, int type, int protocol);
```

Cette fonction prend trois paramètres qui sont liés a l'architecture du réseau utilisé, ainsi le paramètre family peut prendre principalement une des valeurs prédéfinies suivantes :

```
AF_UNIX          protocole Unix interne
AF_INET          protocole Internet
```

Le paramètre type, est lié à la façon dont les paquets transitent. le couple family/type correspond généralement à un protocole de communication particulier, en ce qui nous concerne nous allons utiliser TCP. Avec AF_INET SOCK_STREAM=TCP et SOCK_DGRAM=UDP.

Le dernier paramètre protocole, est un paramètre additionnel, que nous mettrons à 0. Socket renvoie alors un identificateur de socket que l'on utilisera par la suite.

`bind()`

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr *myaddr, int addrlen);
```

Le bind sert a indiquer au sytème la facon dont il doit accepter les connections de la part d'éventels clients, il prend comme paramètre l'identificateur de socket sockfd, obtenu lors de l'ouverture du socket, ainsi qu'une structure de données myaddr spécifique au protocole utilisé contenant les infos nécessaire, ainsi que la taille addrlen de cette structure. Le bind revoie une valeur négative s'il n'a pas pu s'effectuer correctement.

Dans le cas de TCP il faut remplir une structure de type `sockaddr_in`, en initialisant les champs `sin_family, sin_addr.s_addr` et `sin_port`

- sin_family est la famille utilisé, ici AF_INET
- sin_addr.s_addr indique les adresses d'où peuvent être acceptés les requètes de connection, dans notre cas on pourra utiliser la valeur INADDR_ANY. Cependant cette doit être converti au bon format grâce à la fonction `htonl()`
- sin_port est le numéro de port ou doit atterrir la connection, il suffit de choisir un nombre arbitraire plus grand que 1024 (les ports avec des numéros plus petits sont réservés pour des fonctions système). Cette valeur doit être également converti au bon format grâce à la fonction htons().

Les autre champs de la structure doivent être intialisé à zéro (utiliser memset() ou bzero()).

```
bzero((char*)&serv_addr,sizeof(serv_addr));
serv_addr.sinfamily = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(SERV_TCP_PORT);
```

`listen()`

```
#include <sys/types.h>
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

le listen sert à spécifier backlog le nombre de connections simultanées que le serveur peut accepter, la valeur la plus communément employée est 5 qui est la plus grande autorisée. listen revoie une valeur négative en cas de problème.

`accept()`

```
    #include <sys/types.h>
    #include <sys/socket.h>
    int accept(int sockfd, struct sockaddr *peer, int *addrlen);
}
```

Le accept() est un procédure bloquante qui attend qu'une demande de connection arrive, lorsque c'est le cas, une copie du socket est effectuée et son identificateur est renvoyé. Cette manière de faire (copie du socket existant) peut paraître curieuse au premier abortd mais elle est extrèment pratique lorsque l'on écrit un serveur capable d'accepter plusieurs connections simultanées.

Les coordonnées de l'auteur de la requête se trouve dans la structure peer au retour de la fonction accept, de même addrlen permet de connaitre la longueur de la structure en question. Noter que les transactions read/write devront se faire en utilisant l'identificateur de socket renvoyé par accept() et non pas celui renvoyé par socket()

`read()/write()`

Les read/write permettent ensuite de lire/écrire dans le socket. Ils s'utilisent exactement comme ceux des fichiers binaire, si ce n'est qu'ils utilisent un identificateur de socket au lieu d'un identificateur de fichier.

`close()`

```
    int close(int sockfd);
```

Permet de fermer un socket. Il est conseillé de fermer un socket que l'on utilise plus, ainsi on est sûr que le port utilisé est libéré.

3.3 Le client

La partie client est la partie qui effectue demande de connection, elle se programme en ouvrant tout d'abord un socket avec socket() de la même manière que le serveur, il s'agit ensuite d'établir la connection avec connect().

`connect()`

```
    #include <sys/types.h>
    #include <sys/socket.h>
    int connect(int sockfd, struct sockaddr *servaddr, int addrlen);
```

Connect() prend en paramètres, outre l'ID du socket, une structure qui lui indique ou il doit essayer de se connecter, il y a trois valeurs à initialiser dans cette structure de type sockaddr (les autres doivent être mises à zéro).

- sin_family est la famille utilisé, ici AF_INET
- sin_addr.s_addr contient indique l'adresse du serveur, cette adresse est généralement connue sous forme texte (exemple: "129.194.79.80") et doit être convertie en un entier garce à la fonction inet_addr().
- sin_port est le numéro de port du serveur où doit aboutir la connection, il suffit de choisir un nombre arbitraire plus grand que 1024 (les port plus petits sont reservés par le système). Cette valeur doit être également convertie au bon format grace à la fonction htons().

Une fois la connection acceptée, le client n'a plus qu'à lire/écrire dedans et l'aide des fonctions read() et write() puis à le fermer à l'aide de close().

3.4 A vous de jouer !

En vous inspirant de l'exemple précédent, et de la RFC qui se trouve sur `sinfo1` dans `/home/profs/PUBLIC/M6/rfc868.txt` proposer un client qui permet de ce connecter au serveur décrit.

Annexe : les états TCP

- LISTEN : la connexion reste en attente d'une requête de connexion externe par un TCPdistant. Cet état est atteint après une demande de connexion passive.
- SYN-SENT : la connexion se met en attente d'une requête de connexion, après avoir envoyé elle-même une requête à un destinataire.
- SYN-RECEIVED : les deux requêtes de connexion se sont croisées. La connexion attend confirmation de son établissement.
- ESTABLISHED : la connexion a été confirmée de part et d'autre et les données peuvent transiter sur la voie de communication. C'est l'état stable actif de la connexion.
- FIN-WAIT-1 : sur requête de déconnexion émise par l'application, la connexion demande la confirmation d'une requête de déconnexion qu'elle a elle-même émise vers le distant.
- FIN-WAIT-2 : la connexion se met en attente d'une requête de déconnexion par le distant, une fois reçue la confirmation de sa propre requête.
- CLOSE-WAIT : la connexion se met en attente d'une requête de déconnexion émise par l'application.
- CLOSING : la connexion attend la confirmation de sa requête de déconnexion par le TCP distant, lequel avait auparavant émis sa propre requête de déconnexion.
- LAST-ACK : la connexion attend la confirmation de sa requête de déconnexion, émise suite à une requête similaire à l'initiative du distant.
- TIME-WAIT : un temps de latence avant fermeture complète du canal, pour s'assurer que toutes les confirmations ont bien été reçues.
- CLOSED : la connexion n'existe plus. C'est un pseudo état.