Programmation Réseau
Les sockets

Sommaire :

I - Introduction

Le but de la **programmation réseau** est de permettre à des programmes de dialoguer (d'échanger des données) avec d'autres programmes qui se trouvent sur des **ordinateurs distants**, connectés par un **réseau**.

Nous verrons tout d'abord des notions générales telles que les **adresse IP** ou la notion de **port** lié aux **protocoles TCP** ou **UDP**, avant d'étudier les **sockets** Unix/Linux qui permettent à des programmes d'établir une communication et de dialoguer.

II - Adressage

II.1. Adresses MAC et IP

Chaque interface de chaque ordinateur est identifiée par une **adresse** et une **adresse MAC**. On peut voir cette configuration réseau à l'aide de la commande **ip addr** :

```
$ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP group default qlen 1000
    link/ether 80:fa:5b:7b:86:4a brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.25/24 brd 192.168.1.255 scope global dynamic noprefixroute
enp1s0f1
       valid_lft 26359sec preferred_lft 26359sec
    inet6 2a01:e0a:5a0:92a0:d047:62ce:3f09:1dc/64 scope global temporary dynamic
       valid_lft 86029sec preferred_lft 68414sec
    inet6 2a01:e0a:5a0:92a0:ea97:4cbe:ecb3:d90/64 scope global temporary deprecated
dynamic
       valid_lft 86029sec preferred_lft 0sec
```

```
    inet6 2a01:e0a:5a0:92a0:fcef:a571:a3c8:3ac7/64 scope global dynamic mngtmpaddr
noprefixroute
       valid_lft 86029sec preferred_lft 86029sec
    inet6 fe80::3b5f:3301:faaa:da91/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

On voit l'adresse **MAC** de l'interface principale **enp1s0f1** vaut **80:fa:5b:7b:86:4a** et l'adresse **IPv4 192.168.1.25/24**. Cela signifie que le premier octet de l'adresse **IPv4** est égal à **192**, le deuxième **168**, le troisième **1**, et le quatrième vaut **25**.

II.2. Codage d'une adresse IP

Dans un **programme C**, les 4 octets d'une adresse **IP** peuvent être stockés dans un **unsigned long int** (**32 bits**).

On peut stocker toutes les données d'adresse dans une structure **in_addr** :

```
struct in_addr {
        unsigned long int s_addr;
        }
```

On peut traduire l'adresse IP (**s_addr**) en une chaîne de caractère (avec les octets écrits en décimal et séparés par des points, exemple : "**192.168.0.2**") par la fonction **inet_ntoa()** :

```
char * inet_ntoa(struct in_addr adresse);
```

Remarque : La chaîne de caractères renvoyée par **inet_ntoa()** contient les octets en partant du poids faible et non du poids fort. L'adresse "**192.168.0.2**" correspond à la chaîne de caractères "**2.0.168.192**".

Inversement, on peut traduire une chaîne de caractère représentant une adresse IP en **struct in_addr**, en passant la structure par adresse à la fonction **inet_aton()** :

```
int inet_aton(const char *chaine, struct in_addr *adresse);
```

Exemple de programme :

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int main(void)
{
unsigned long int add = 256;
struct in_addr addr;
char str[15];
sprintf(str, "%lu", add);
inet_aton(str, &addr);
printf("Valeur entière initiale : %lu\n", add);
printf("IP (valeur entière) : %u\n", addr.s_addr);
printf("IP (décimal pointé) : %s\n", inet_ntoa(addr));
return 0;
}
```

Résultat :

```
jcabianca@srv-scribe:~$ ./test
Valeur entière initiale : 256
IP (valeur entière) : 65536
IP (décimal pointé) : 0.0.1.0
```

III - Ports UDP ou TCP

Il peut y avoir de nombreuses applications réseau qui s'exécutent sur la même machine. Les **numéros de port** permettent de préciser avec quel programme nous souhaitons dialoguer par le réseau. **Chaque application** qui souhaite utiliser les services de la **couche IP** se voit attribuer un **numéro de port**.

Un **numéro de port** est un entier codé sur **16 bits** (deux octets). Dans un programme **C**, on peut stocker un numéro de port dans un **unsigned short int**.

Il y a un certain nombre de ports qui sont réservés à des services standards. Les numéros de port inférieurs à **1024** sont réservés aux serveurs et démons lancés par root (éventuellement au démarrage de l'ordinateur), tels que le serveur **ssh** (/usr/sbin/sshd) sur le port **22**.

Pour connaître le numéro de port correspondant à un service tel que **ssh**, on peut regarder dans le fichier **/etc/services** :

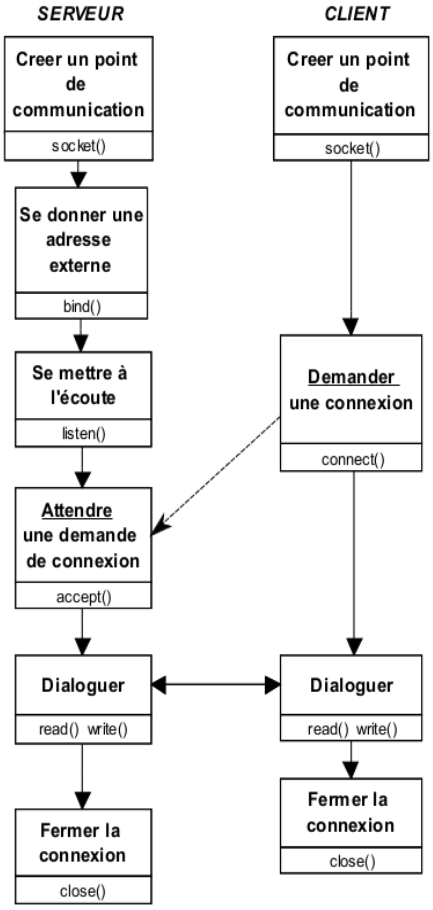
```
ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp
telnet          23/tcp
smtp            25/tcp          mail
bootps          67/tcp                          # BOOTP server
bootps          67/udp
bootpc          68/tcp                          # BOOTP client
bootpc          68/udp
http            80/tcp          www             # WorldWideWeb HTTP
http            80/udp                          # HyperText Transfer Protocol
pop3            110/tcp         pop-3           # POP version 3
pop3            110/udp         pop-3
sftp            115/tcp
https           443/tcp                         # http protocol over TLS/SSL
https           443/udp
```

IV - Les sockets

IV.1. Présentation

Un **socket** est un **dispositif de communication** entre **processus** permettant une communication de type **client/serveur**, soit localement, soit à travers un réseau.

On peut résumer le fonctionnement d'une communication **client/serveur** en mode **connecté** (**TCP**) par le schéma suivant :



Comme dans le cas de l'ouverture d'un fichier, la communication par **socket** utilise un **descripteur** pour désigner la connexion sur laquelle on envoie ou reçoit les données.

L'ouverture d'un **socket** (coté **serveur**) se fait en deux étapes :

- La première opération à effectuer consiste à appeler la fonction ***socket()*** créant un socket et retournant un descripteur (un entier) identifiant de manière unique la connexion ;
- La deuxième opération à effectuer consiste à appeler la fonction ***bind()*** permettant de spécifier le type de communication associé au socket (protocole TCP ou UDP).

IV.2. Exemple de programmes client et serveur

Le programme client :

```
#include <fcntl.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define IP_SERVER       "127.0.0.1"
#define PORT_SERVER     2500
#define TAILLE_MAX      255
```

```
int main(int argc, char**argv)
{
int s;   // descripteur du socket
struct sockaddr_in to;          // structure d'adresses famille AF_INET
char buffer[255];       // buffer de réception

s = socket( AF_INET, SOCK_STREAM, 0);    // Création point de communication
if(s <0 ) // erreur socket
        {
        return(-1);
        }
else // socket OK
        {
        to.sin_family = AF_INET;
        to.sin_addr.s_addr = (inet_addr(IP_SERVER));
        to.sin_port= htons(PORT_SERVER);
        if( connect( s,(struct sockaddr*)&to, sizeof(to)) < 0 )
                {
                printf ("Erreur connexion\n");
                return(-1);
                }
        else
                {
                printf ("Connexion au serveur OK\n");
                int ret = recv( s, buffer, TAILLE_MAX, 0);
                buffer[ret] = '\0';
                printf ("Trame reçue : %s\n", buffer);
                close (s);
                return 0;
                }
        }
}
```

Le programme serveur :

```
#include <fcntl.h>
#include <unistd.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT_SERVER        2500
#define TAILLE_MAX         255

/*****************************************************************/
// main()
/*****************************************************************/
int main(int argc, char**argv)
{
int canal_demande;  // descripteur de connexion principal
int canal_dedie;        // descripteur de connexion secondaire
struct sockaddr_in nom_producteur; // structure d'adresses famille AF_INET

char trame[TAILLE_MAX] = "$GPGGA,,,,,,0,00,99.99,,,,,,*48\r\n"; // buffer d'envoi
```

```
canal_demande = socket( AF_INET, SOCK_STREAM, 0);   // Protocole TCP
if(canal_demande <0 ) // erreur socket
        {
        return(-1);
        }
else // socket OK
        {
        /* Remplissage Structure */
        nom_producteur.sin_family = AF_INET;
        nom_producteur.sin_addr.s_addr=htonl(INADDR_ANY);
        nom_producteur.sin_port= htons(PORT_SERVER);
        /* Association de l'adresse au socket */
        if ( bind(canal_demande, (struct sockaddr *)&nom_producteur,
sizeof(nom_producteur)))
                {
                printf("Erreur de bind\n");
                }
        else
                {
                listen(canal_demande, 5);    //Fixe la longueur de la file des appels a 5
                printf ("Attente connexion client\n");
                canal_dedie = accept(canal_demande, 0, 0);
                int ret = send(canal_dedie,trame,strlen(trame), 0);
                if (ret != 0) printf ("Trame envoyée : %s\n", trame);
                else printf ("Erreur envoi trame\n");
                sleep(2);
                close (canal_dedie);
                return 0;
                }
        }
}
```

Résultats :

```
jcabianca@srv-scribe:~$ ./serveur
Attente connexion client
Trame envoyée : $GPGGA,,,,,,0,00,99.99,,,,,,*48

jcabianca@srv-scribe:~$ ./client
Connexion au serveur OK
Trame reçue : $GPGGA,,,,,,0,00,99.99,,,,,,*48
```

IV.3. Exemple de client avec Qt

Exemple de création d'un socket client en mode synchrone :

```
// .....
#include <Qdebug>
#include <iostream>

QString addr="127.0.0.1";
quint16 port=1024;
int taille=0;
char trame[83];

QTcpSocket *client;
```

```
client = new QTcpSocket();

client->connectToHost(addr, port);
if (client->waitForConnected(1000))  // Attente de connexion au maximum 1s
      {
      // Lecture d'un message en provenance du serveur
      while(client->bytesAvailable() == 0) client->waitForReadyRead();
      taille = client->bytesAvailable();
      client->read(trame, taille);
      trame[taille] = '\0';
      std::cout << "Trame : " << trame << std::endl;
      }
else std::cout << "Connexion au serveur impossible" << " \n";
client->close();
// .......
```