Programmation réseau

Module FAR

Polytech Montpellier – IG3

David Delahaye

Utilisation des sockets

- Apparues en 1984, lors de la création des distributions BSD.
- Points de terminaison mis à l'écoute sur le réseau, afin de faire transiter des données logicielles.
- Associées à une adresse IP et un numéro de port.
- Associées à un protocole (UDP ou TCP).
- Utilisation différente suivant le système ; on les utilisera en Linux, « what else? » ;-).

Sockets = canaux d'échange entre 2 processus (machines) distants
c'est bidirectionnel

Création des sockets (En C)

- Déclaration de la socket :

  SOCKET sock;

Udp = pas en mode connecte (envoi paquet sans demande de retour). On sait pas si ca va arriver et dans quel ordre...

TCP = avec réponse. (Qd on veut envoyer des donnees correctement)

- Création de la socket :

  int socket(int domain, int type, int protocol);

  - domain = AF_INET (pour TCP/UDP) ou AF_UNIX pour des communications en local sur une machine Unix.
  - type = SOCK_STREAM (pour TCP) ou SOCK_DGRAM (pour UDP).
  - protocol = 0 pour TCP/UDP.

Configuration des sockets

Il faut donner un contexte à la stocket après création.

- struct sockaddr_in

```
{
   short      sin_family;
   unsigned short   sin_port;
   struct   in_addr   sin_addr;
   char   sin_zero[8];
};
```

Configuration des sockets

- sin_family = AF_INET.
- sin_port = port rendu par la fonction hton avec comme paramètre le port utilisé.
- sin_addr.s_addr = adresse IP.

Fonctions déjà définies :

  - inet_addr("127.0.0.1")
  - htonl(INADDR_ANY) (IP de la machine où l'application s'exécute)
  - const char *hostname = "www.google.fr";

    hostinfo = gethostbyname(hostname); Appel au DNS

    sin.sin_addr = *(in_addr *) hostinfo->h_addr ;

struct

Associer une socket à sa configuration

- int bind(int socket, const struct sockaddr* addr, socklen_t addrlen);

- socket = la socket déclarée et crée.
- addr = la structure de type sockaddr_in construite. Pas le même type, on fait un « cast » (ha ha ha, c'est du C).
- addrlen = taille de la structure de configuration (utilisation de sizeof).
- Exemple : bind(sock, (sockaddr*)&sin, sizeof(sin));

Mettre une socket en écoute

- int listen(int socket, int backlog);

- Protocole TCP (mode connecté).
- socket = la socket qui va être utilisée.
- backlog = représente le nombre maximal de connexions pouvant être mises en attente.
- Exemple : listen(sock, 5); Au total, j'accepte simultanément d'avoir 5 connections maximum.

Établir une connexion côté serveur

- int accept(int socket, struct sockaddr* addr, socklen_t* addrlen);
- Protocole TCP (mode connecté).
- Retourne la socket du client qui se connecte au serveur.
- socket = la socket serveur utilisée.
- addr = configuration de la socket client (elle sera remplie lors d'une connexion).
- addrlen = taille de la configuration de la socket client.

Donne a la fin une socket CLIENT

```
socklen_t taille = sizeof(csin);
csock = accept(sock, (sockaddr*)&csin, &taille);
```

Cote serveur : il y a 2 sockets (client et celui qui permet de discuter avec client)

Jamais utiliser SOCK pour discuter car sinon on envoie infos a nous meme !!

Établir une connexion côté client

- int connect(int socket, struct sockaddr* addr, socklen_t addrlen);

- Protocole TCP (mode connecté).
- Retourne la socket du client qui se connecte au serveur.
- socket = la socket client utilisée.
- addr = configuration de la socket serveur.
- addrlen = taille de la configuration de la socket client (utilisation d'un sizeof).

Fermer une socket

Toujours fermer quand on sort

- int close(int socket) ;

- socket = la socket à fermer..

Exemple d'une connexion TCP

- Côté serveur :

Serveur = Écoute sur un port (cad qu'il attend sur un port et des qu'une socket veut se connecter sur ce port, le serveur accepte de se connecter avec et utilise donc Client)

Client = pour pouvoir parler avec la socket qui s'est connectée

```
/* Socket et contexte d'adressage du serveur */
   sockaddr_in sin;
   socket sock;
   socklen_t recsize = sizeof(sin);

   /* Socket et contexte d'adressage du client */
   sockaddr_in csin;
   socket csock;
   socklen_t crecsize = sizeof(csin);
```

Sin = adresse ip et port de socket serveur
Csin = contexte de la socket client

Exemple d'une connexion TCP

```
sock = socket(AF_INET, SOCK_STREAM, 0);
sin.sin_addr.s_addr = htonl(INADDR_ANY);  /* Adresse IP automatique */
sin.sin_family = AF_INET;                 /* Protocole familial (IP) */
sin.sin_port = htons(PORT);               /* Listage du port */
bind(sock, (sockaddr*)&sin, recsize);
listen(sock, 5);
csock = accept(sock, (sockaddr*)&csin, &crecsize);
closesocket(csock);
closesocket(sock);
```

Port >1024

Exemple d'une connexion TCP

- Côté client :

Attention : deux programmes différents (cote client et serveur). Il faut compiler les 2 puis on pourra établir connection.

```
sock = socket(AF_INET, SOCK_STREAM, 0);
sin.sin_addr.s_addr = inet_addr("127.0.0.1");
sin.sin_family = AF_INET;
sin.sin_port = htons(PORT);
connect(sock, (sockaddr*)&sin, sizeof(sin)) ;
close(sock) ;
```

Attention : il faut connaître le port du serveur pour pouvoir se connecter.

Transmission de données (envoi)

- int send(int socket, void* buffer, size_t len, int flags);

- socket = socket destinée à recevoir le message.
- buffer = buffer de données.
- len = nombre d'octets à envoyer.
- flags = type d'envoi (ici toujours à 0).
- Exemple :

  char buffer[32] = "Bonjour !";

  send(sock, buffer, 32, 0);

Transmission de données (réception)

- int recv(int socket, void* buffer, size_t len, int flags);

- socket = socket destinée à attendre le message.
- buffer = buffer de données.
- len = nombre d'octets à envoyer.
- flags = type d'envoi (ici toujours à 0).
- Exemple :

  char buffer[32] = ""; Buffet de réception

  recv(sock, buffer, 32, 0);

Si rcv = -1 alors erreur
0 alors rien reçu
>0 alors = nb octets reçu

Attention : receive est bloquant. C'est pourquoi il faut fermer les sockets quand on ne veut plus l'utiliser.

Transmission de données (fermeture)

- int shutdown(int socket, int how);

Shutdown : Ferme la socket au niveau des flux d'entrée/sortie (d'un point de vue écriture lecture). Donc la socket existe toujours ≠ close qui ferme complètement la socket (et donc supprime socket)
On fait donc d'abord le shutdown avant le close.

- socket = socket dont on doit fermer la connexion.
- how = comment on ferme la connexion :
  - 0 : fermeture en réception ;
  - 1 : fermeture en émission ;
  - 2 : fermeture dans les deux sens.

TP sockets : exercice n°1

- Récupérer les fichiers C sous Moodle
  - Fichier « tcp_serveur.c » : serveur TCP ;
  - Fichier « tcp_client.c » : client TCP ;
  - Makefile (pour compiler automatiquement).
- Compiler avec make.
- Exécuter les programmes :
  - Comment faut-il les exécuter ?
  - Peut-on exécuter plusieurs clients ?
  - Peut-on exécuter plusieurs serveurs ?

TP sockets : exercice n°2

- Copier les fichiers précédents dans un autre répertoire.
- Modifier les deux fichiers de manière à ce que :
  - Une fois un client connecté, le serveur envoie « Bonjour ! » au client (le client affichera ce qu'il a reçu) ;
  - Une fois le message reçu, le client enverra « Merci ! » au serveur (le serveur affichera ce qu'il a reçu) .
- Tester indépendamment :
  - Son serveur avec le client du groupe d'à côté ;
  - Son client avec le serveur du groupe d'à côté. ;
  - Que faut-il modifier ?

TP sockets : exercice n°3

- Copier les fichiers précédents dans un autre répertoire (revenir à des sockets en local).
- Modifier les deux fichiers de manière à ce que :
  - Le serveur puisse accepter en boucle des connexions.
  - Quel problème entrevoit-on ?

TP sockets : exercice n°4

- Copier les fichiers précédents dans un autre répertoire.
- Modifier les deux fichiers de manière à ce que :
  - Le serveur puisse accepter en boucle des connexions mais crée un nouveau processus fils (« fork ») pour gérer la connexion avec le client, le processus père continuant d'attendre d'autres connexions clients ;
  - Le client demande la saisie d'une chaîne de caractères à l'utilisateur (utiliser la fonction « fgets »), l'envoie au serveur (qui affichera ce qu'il a reçu), puis sorte.

TP sockets : exercice n°5

- Copier les fichiers précédents dans un autre répertoire.
- Modifier les deux fichiers de manière à ce que :
  - On ait le même comportement que précédemment sauf que le client demande en boucle (infinie) des messages qu'il envoie au serveur.
  - On rattrape le Ctrl-C dans le client de manière à ce que la socket soit convenablement fermée avant de quitter.

TP sockets : exercice n°6

- Copier les fichiers précédents dans un autre répertoire.
- Modifier les deux fichiers de manière à ce que :
  - On utilise une « thread » plutôt qu'un « fork ». Qu'est-ce qui change ? À quoi doit-on faire attention ?