UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

Les sockets

Mode connecté (TCP)

Cet article présente comment créer un client et un serveur TCP et envoyer et lire une chaîne de caractères via le réseau.

1 Le programme client

Dans premier temps, le client doit créer une socket en mode connecté à l'aide de l'appel à `socket`. Le mode et le protocole sont passés en paramètres à l'aide de constantes.

```
int sockfd;
if((fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) {
  perror("Erreur lors de la création de la socket ");
  exit(EXIT_FAILURE);
}
```

Le client doit se connecter au serveur. Il faut initialiser une structure `sockaddr_in` qui correspond à l'adresse du serveur. Nous devons remplir les trois champs suivants :

- `sin_family` : la famille de protocoles (ici `AF_INET` pour IPv4)
- `sin_port` : le port (ne pas oublier de le mettre dans le bon format à l'aide de `htons`)
- `sin_addr` : l'adresse (nous utilisons `inet_pton` pour convertir la chaîne de caractères contenant l'adresse dans le bon format)

Il est conseillé de "vider" la structure avant, avec la fonction `memset`.

```
struct sockaddr_in adresse;
memset(&adresse, 0, sizeof(struct sockaddr_in));
adresse.sin_family = AF_INET;
adresse.sin_port = htons(atoi(argv[2]));
if(inet_pton(AF_INET, argv[1], &adresse.sin_addr) != 1) {
  perror("Erreur lors de la conversion de l'adresse ");
  exit(EXIT_FAILURE);
}
```

Le client peut maintenant se connecter au serveur à l'aide de `connect`.

```
if(connect(fd, (struct sockaddr*)&adresse, sizeof(adresse)) == -1) {
  perror("Erreur lors de la connexion ");
  exit(EXIT_FAILURE);
}
```

La socket est maintenant prête à être utilisée pour l'envoi et la réception de messages. À l'aide de `write`, nous envoyons d'abord la taille du message puis le message en lui-même.

```
size_t taille = strlen(argv[3]) + 1;
if(write(fd, &taille, sizeof(size_t)) == -1) {
  perror("Erreur lors de l'envoi de la taille du message ");
  exit(EXIT_FAILURE);
}
if(write(fd, argv[3], sizeof(char) * taille) == -1) {
  perror("Erreur lors de l'envoi du message ");
  exit(EXIT_FAILURE);
}
printf("Client : message envoyé.\n");
```

2 Le serveur

Le serveur doit lui-aussi créer sa socket avec le même appel que le client.

```
int sockfd;
if((sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) {
  perror("Erreur lors de la création de la socket ");
  exit(EXIT_FAILURE);
}
```

Il faut ensuite la nommer à l'aide de `bind`. Comme pour le client, nous devons initialiser une structure `sockaddr_in` mais l'adresse IP n'est pas spécifiée (on laisse le système choisir une adresse active à l'aide de la constante `INADDR_ANY`). Le numéro de port est passé en argument.

```
struct sockaddr_in adresse;
memset(&adresse, 0, sizeof(struct sockaddr_in));
adresse.sin_family = AF_INET;
adresse.sin_port = htons(atoi(argv[1]));
adresse.sin_addr.s_addr = htonl(INADDR_ANY);
```

Puis on nomme la socket à l'aide de `bind` :

```
if(bind(sockfd, (struct sockaddr*)&adresse,
        sizeof(struct sockaddr_in)) == -1) {
  perror("Erreur lors du nommage de la socket ");
  exit(EXIT_FAILURE);
}
```

Nous plaçons maintenant la socket en mode passif, c'est-à-dire que nous indiquons qu'elle est utilisée pour recevoir des connexions (on dit que c'est une socket de connexion).

```
if(listen(fd, 1) == -1) {
  perror("Erreur lors de la mise en mode passif ");
  exit(EXIT_FAILURE);
}
```

Le serveur se met en attente d'une connexion à l'aide de `accept` qui retourne une nouvelle socket dite de communication :

```
int sockClient;
printf("Serveur : attente de connexion...\n");
if((sockclient = accept(fd, NULL, NULL)) == -1) {
  perror("Erreur lors de la demande de connexion ");
  exit(EXIT_FAILURE);
}
```

Une fois la connexion établie, le serveur est prêt à recevoir le message (la taille puis le message) :

```
char *msg;
size_t taille;
if(read(sockclient, &taille, sizeof(size_t)) == -1) {
  perror("Erreur lors de la lecture de la taille du message ");
  exit(EXIT_FAILURE);
}
if((msg = (char*)malloc(sizeof(char) * taille)) == NULL) {
  perror("Erreur lors de l'allocation mémoire pour le message ");
  exit(EXIT_FAILURE);
}
if(read(sockclient, msg, sizeof(char) * taille) == -1) {
  perror("Erreur lors de la lecture de la taille du message ");
  exit(EXIT_FAILURE);
}
printf("Serveur : message recu '%s'.\n", msg);
```

3 Compilation et exécution

Le `makefile` fourni permet de compiler les programmes précédents. Saisissez la commande suivante :

```
make
```

Pour tester les programmes, dans un premier terminal, exécutez le serveur (ici, nous choisissons le port 12000) :

```
./serveur 12000
```

Dans un second terminal (ou dans un nouvel onglet), exécutez le client :

```
./client 127.0.0.1 12000 "Bonjour␣tout␣le␣monde"
```