Les sockets

Mode non connecté (UDP)

Cet article présente comment créer un client et un serveur UDP 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 non 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((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
  perror("Erreur lors de la création de la socket ");
  exit(EXIT_FAILURE);
}
```

La socket est maintenant prête pour envoyer des messages. Dans notre cas, nous allons envoyer une chaîne de caractères de taille fixée. Il faut d'abord initialiser une structure `sockaddr_in` qui correspond à l'adresse du destinataire du message. 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 adresseServeur;
memset(&adresseServeur, 0, sizeof(struct sockaddr_in));
adresseServeur.sin_family = AF_INET;
adresseServeur.sin_port = htons(atoi(argv[2]));
if(inet_pton(AF_INET, argv[1], &adresseServeur.sin_addr) != 1) {
  perror("Erreur lors de la conversion de l'adresse ");
  exit(EXIT_FAILURE);
}
```

Nous récupérons le message passé en argument que nous plaçons dans une chaîne de caractères de taille fixée (la constante `TAILLE_MAX` est définie dans le fichier `include.h`).

```
char msg[TAILLE_MAX];
memset(msg, 0, TAILLE_MAX);
strcpy(msg, argv[3]);
```

La socket et le message sont prêts. Il ne reste plus qu'à envoyer le message à l'aide de `sendto`.

```
if(sendto(sockfd, &msg, sizeof(msg), 0,
          (struct sockaddr*)&adresseServeur,
          sizeof(struct sockaddr_in)) == -1) {
  perror("Erreur lors de l'envoi du message ");
  exit(EXIT_FAILURE);
}
```

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_DGRAM, IPPROTO_UDP)) == -1) {
  perror("Erreur lors de la création de la socket ");
  exit(EXIT_FAILURE);
}
```

Par contre, pour recevoir des messages, il faut 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 adresseServeur;
memset(&adresseServeur, 0, sizeof(struct sockaddr_in));
adresseServeur.sin_family = AF_INET;
adresseServeur.sin_port = htons(atoi(argv[1]));
adresseServeur.sin_addr.s_addr = htonl(INADDR_ANY);
```

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

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

Le serveur est prêt à recevoir le message :

```
char msg[TAILLE_MAX];
printf("Serveur en attente du message du client.\n");
if(recvfrom(sockfd, &msg, sizeof(msg), 0, NULL, NULL) == -1) {
  perror("Erreur lors de la réception 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"
```