TP2 – **Client - Serveur echo**

Yann.Esposito@lif.univ-mrs.fr

14 octobre 2005

☞ Dans ce TP nous programmerons un client et un serveur utilisant le protocole TCP/IP. Nous utiliserons du C mais aussi pour des raisons de difficulté à venir, nous introduirons du Python.

1 Généralités sur TCP/IP

Alors que le rôle de IP est d'envoyer des trames d'un ordinateur vers un autre. Le rôle de TCP est d'en assurer l'envoie dans le bon ordre.

Les programmes utilisant TCP/IP sont formé de la sorte : coté client :

```
Création d'une socket TCP/IP.
Connection de cette socket avec une destination (adresse IP + port TCP)
Communication entre le client et le serveur :
    envoie et réceptions de messages
fermeture de la socket
```

coté serveur :

```
Création d'une socket TCP/IP.
Lier la socket à un port TCP
Écouter sur la socket
pour toujours:
    soit la nouvelle socket et l'adresse du client qui se connectent (accept)
    prise en charge du client (réception et envoie de données)
```

Les socket

La création d'une socket se fait en utilisant la fonction `socket`. Cette fonction prend trois arguments : le *domaine*, le *type* et le *protocole*.

- Le domaine spécifie le domaine de communication qui va être utilisé. Cet argument permet de sélectionner la famille de protocoles qui sera utilisé. Le domaine qui va nous intéresser ici sera `AF_INET` qui correspond aux protocoles internet (ARPA).
- Le type va définir la sémantique des communications, celle qui va nous intéresser sera `SOCK_STREAM`. Ce type permet de donner une connection en deux direction, par séquences avec des flux.
- Le protocole spécifie le protocole particulier qui sera utilisé avec la socket (traduction de socket). Pour cet exercice nous utiliserons le protocole `IPPROTO_TCP`.

Les socket doivent établir une connection avant de pouvoir être utilisées comme des flux de données. Les socket permettent de s'assurer qu'il n'y aura ni perte ni duplication des données. Tout se passe comme si on lisait un fichier.

la fonction socket renvoie -1 si une erreur s'est produite. Dans le cas contraire, elle renvoie un descripteur de socket (comme un descripteur de fichier mais avec une socket).

Pour résumer :

```
/* Les arguments de socket correspondent au type TCP */
if ((s=socket(PF_INET,SOCK_STREAM,0))<0) {
   perror ("Impossible d'allouer la chausette !");
   exit(-1);
}
```

La structure sockaddr

Une fois que vous avez créer votre socket, vous devez la connecter à quelqu'un en utilisant la commande `connect`. `connect` prend trois paramètres : la socket, un pointeur vers une structure `sockaddr` et la taille de la structure.

voici comment se fait une construction de `sockaddr` habituelle :

```
struct sockaddr_in leserver;
...
memset(&leserver, 0, sizeof(leserver)); // initialisation de la structure
echoserver.sin_family = AF_INET;        // Internet/IP
echoserver.sin_addr.s_addr = inet_addr("147.94.66.131"); // Addresse IP
echoserver.sin_port = htons(2001);      // le port
```

Il suffit ensuite d'écrire :

```
if (connect(sock, (struct sockaddr *) & leserver, sizeof(echoserver)) < 0) {
   fprintf(stderr,"Impossible de se connecter au serveur\n");
   exit(-1);
}
```

1.1. Écrivez une fonction `monConnect(socket sock, const char *adresse, const int port)` qui connecte aussi la socket.

le dialogue

Maintenant que vous avez réussi à connecté votre socket jusqu'à la bonne personne, il ne vous reste plus qu'à discuter avec votre interlocuteur. Pour cela, il suffit d'utiliser les fonctions `send` et `recv`. Ces commandes s'utilisent de manière similaire aux fonctions d'écritures et de lecture sur fichier.

Lorsque vous avez fini de parler ou de recevoir des informations, il ne vous reste plus qu'à fermer votre socket avec `close()`.

2 Serveur C

La construction du serveur est très proche de celle du client, les différences se situent lors de la création de la structure `sockaddr`. De plus un serveur doit se lier la socket et doit écouter.

Dans cet exercie, utilisez :

```
struct sockaddr_in echoserver;
...
echoserver.sin_addr.s_addr = htonl(INADDR_ANY); /* addresse entrante */
...
```

Une fois que la structure est correctement définie, il faut lier la socket. C'est-à-dire lui donner un nom qui correspond au contenu de la structure sockaddr. Pour cela, on utilise `bind`.

Une fois liée, on doit écouter dans la socket. Pour cela, on utilise la fonction `listen`.

Puis généralement, le serveur entre dans une boucle infinie dans laquelle il accepte les connections avec la fonction `accept` et traite la nouvelle connection.

2.1. En utilisant les manuels de `bind`, `liste`, `accept`, `send` et `recv`, programmez un serveur echo dont le but et d'attendre la connection d'un client et de lui renvoyer tout ce que le client lui envoie.

2.2. Vérifiez le fonctionnement de votre serveur en utilisant la commande `telnet`.

3 Client en C

3.1. En utilisant les informations contenues dans cette planche de TP ainsi que des manuels de `htons`, `send`, `recv` et de `close` écrivez un client qui envoie une phrase à un serveur qui est censé lui renvoyé tout ce qu'il a reçut.

Appelez votre programme echoclient. On doit pouvoir l'utiliser de la manière suivante :
`echoclient <mot> <ip du serveur> <port>`

3.2. programmez le client de façon à ce qu'il se comporte comme un terminal ; c'est-à-dire qu'au début il vous demande d'écrire quelque chose (avec un prompt >). Puis lorsqu'un mot est tapé, il envoie ce mot au serveur et écrit ce qu'il reçoit. Puis il vous redonne la main. De plus pour vous aidez, vous avez acces à un exemple de serveur commenté sur le site http://www.cmi.univ-mrs.fr/~esposito.

4 Client/Serveur en Python

☞ Le langage Python est un langage de script récent qui permet d'arriver au même résultat que les autres langages de programmation en beaucoup moins de ligne de code.
Contrairement au `C`, Python cache beaucoup de choses au programmeur. Cette façon de faire permet de coder plus vite des programmes compliqués au détriment d'une certaine forme de compréhension. Aussi, comme après un certain labs de temps les aspect très proches de la machines seront un frein à l'évolution des TP, ceux-ci vont de manière progressive se faire de plus en plus en Python.
Les TP de réseaux n'ont en aucune manière l'intention de vous apprendre le Python. Néamoins, il me semble que toute personne sachant programmer en C ou en Java doit pouvoir apprendre à programmer en Python en quelques heures. Pour cela, je vous demande de faire entièrement le tutoriel situé à l'adresse suivante :
`http://www.python.org/doc/current/tut/tut.html`
Prenez le temps qu'il faut pour intégrer tous les aspets. Normalement, il suffit d'environ 2 à 3 heure pour finir le tutoriel et connaître assez de Python pour pouvoir faire le travail qui vous sera demandé par la suite.
Considération importante : Comme dans la plupart des langages récents, en Python, toutes les variables sont des pointeurs vers des objets à l'exception des types de bases (entiers et charactères).
Dans les TP qui nous intéresseront, les notions de programmation objets seront quasi-inutiles. Ne perdez pas de temps à comprendre les notions de classes si vous n'êtes pas familier avec elles.

Plutôt que vous demander de refaire le programme en Python, allez voir sur mon site des solutions possibles :

http://www.cmi.univ-mrs.fr/~esposito/enseignement/enseignement.php

Notez que même si ces programmes sont plus courts il peuvent faire plus de choses que le programme en `C` que vous deviez faire. En effet, plutôt qu'utiliser une adresse IP, on peut utiliser un nom. La résolution par DNS est intégrée.

La semaine prochaine il vous sera demandé d'arriver vous-même à faire des programmes en Python.

5 Ethereal

5.1. lancez `ethereal` et regardez les paquets qui passent sur le réseauxlorsque vous utilisez votre couple client/serveur.