TD-TP 11

Socket et Client

Chaque machine (*hôte*) d'un réseau informatique doit posséder une adresse la caractérisant. Par exemple, `192.168.60.5` est une adresse IP (réseau internet).

Pour faciliter la gestion et l'utilisation des machines, on peut associer des adresses symboliques (noms) aux adresses IP. Par exemple, sur internet, `www.univ-paris13.fr` est le nom d'une machine ayant l'adresse `194.254.164.240`. En salles de TP, les noms F204-2, etc. correspondent aux adresses des différentes machines. Ces adresses sont privées.

De plus, chaque machine dispose de *ports* logiques (un numéro entre 1 et 65535) permettant de séparer les informations associées aux différents services offerts (messagerie, web, etc).

Une *socket* (ou « prise ») est un point de communication permettant à un processus d'émettre et de recevoir des informations. Une *socket* est identifiée par un entier, elle s'utilise comme un descripteur de fichier. La communication dans une *socket* s'effectue dans les deux sens contrairement aux tubes. En TCP, pour que cette communication puisse avoir lieu il faut que la *socket* soit connectée à une autre socket.

1 En TD

Exercice 1 (Client TCP/IP)**.** *L'objectif de cet exercice est d'écrire le code C d'un client TCP* `Client` *qui sera utilisé en TP. On s'aidera de l'annexe.*

1. *Dans le modèle client/serveur, quel processus demande la connection ? Quelle fonction C permet de réaliser cette étape ?*
2. *Que doit-on avoir crée avant la connexion, à la fois coté serveur et coté client ?*
3. *Comment le client dialogue t-il avec le serveur ?*
4. *Quelle instruction met fin à la connexion ?*

Le nom et le numéro de port du serveur seront passés au client en paramètres, à la ligne de commande. Ainsi la ligne de commande `Client www.univ-paris13.fr 80` *établira une connexion avec la machine* `www.univ-paris13.fr`*, sur le port 80.*

5. *Quelle opération doit on effectuer pour obtenir l'adresse de connexion ?*
6. *En déduire la structure du programme client et commencer à la remplir, en laissant vide la fonction de dialogue.*
7. *Quel paramètre devra t-on passer à la fonction de dialogue ?*

Pour la fonction de dialogue, on supposera que le client et le serveur envoient tour à tour une ligne de texte, à commencer par le client. Le texte envoyé par le client sera saisi par l'utilisateur, et la réponse du serveur sera affichée en dessous.

8. *Compléter votre code.*
9. Pour chez vous. *Modifier le programme de manière à ce que le client termine la connexion lorsque l'utilisateur envoie le texte* `quit`*.*

Exercice 2 (Pour chez vous)**.** *Rappeler les différentes étapes d'une connexion client/serveur TCP/IP, du coté client et du coté serveur, et les fonctions C utilisées (se reporter au cours).*

2 En TP

Exercice 3 (*sockets* en ligne de commande)**.** *L'utilitaire libre* `socket` *permet de créer des clients et des serveurs en ligne de commande. Cet utilitaire n'est pas installé de manière standard sur les machines de TP, toutefois vous le trouverez dans le répertoire* `~boudes/bin/`*. Sa page de manuel (en anglais) peut être consultée par la commande* `man ~boudes/man/man1/socket.1.gz`

1. *Quel est le nom de votre machine ?*
2. *Que font les commandes suivantes :*

   ```
   ~boudes/bin/socket -slf 7777
   ~boudes/bin/socket nom_de_la_machine 7777
   ~boudes/bin/socket -slfp "~boudes/bin/freud" 7778
   ```

   Les tester dans des terminaux différents, puis sur des machines différentes.
3. *Peut on connecter plusieurs clients au même serveur ? À qui vont les réponses tapées dans le terminal du serveur ?*
4. Pour chez vous. *La page de* `man` *de* `socket` *vous dit ce que font les options* `-l, -f` *(boucle et fork). Mais à quoi servent elles concrètement ?*

Exercice 4 (Tester le client du TD)**.**

1. *Taper, compiler et essayer votre programme client.*
   - *Avec un serveur crée par la commande* `socket`
   - *Avec comme serveur, le programme* `~/boudes/bin/Serveur`
2. *Que se passe t-il lorsque votre client envoie le message* `psy` *à ce dernier serveur ?*
3. Pour chez vous. *À votre avis comment fait on pour programmer cette fonctionnalité ?*

3 À suivre...

Le prochain TD sera consacré à l'écriture d'un serveur TCP, fonctionnant comme `Serveur`. Pour le préparer, vous pouvez traiter les questions et exercices marqués « pour chez vous » ci-dessus.

Annexe : Structures et fonctions C

1. Récupération des caractéristiques d'une machine distante à partir de son nom :

   ```
   #include <netdb.h>
   struct hostent *gethostbyname( /* renvoie NULL si erreur */
           const char *nom /* nom de la machine */
   );

   struct  hostent {
           char    *h_name;        /* nom officiel de la machine */
           char    **h_aliases;    /* liste d'alias */
           int     h_addrtype;     /* type d'adresse (AF_INET) */
           int     h_length;       /* longueur de l'adresse */
           char    **h_addr_list;  /* liste des adresses, ordre du réseau */
   #define h_addr  h_addr_list[0]  /* premiere adresse */
   };
   ```

2. Création d'une *socket* :

```
#include <sys/types.h>
#include <sys/socket.h> /* socket(), bind() ... */
int socket(
    int domaine,  /* AF_UNIX | AF_INET *//* AF_INET <-- IP */
    int type,     /* SOCK_DGRAM | SOCK_STREAM *//* SOCK_STREAM <-- TCP */
    int protocole /* 0 : protocole par defaut */
);
```

3. Supprimer une *socket* : `close(int sock)`.
4. Débuter une connexion entre *sockets* :

```
int connect(
    int                    socket, /* descripteur de la socket du client */
    const struct sockaddr *addr,   /* adresse de la socket du serveur */
    size_t                 l_addr  /* taille de cette adresse  */
);
```

   Si la *socket* est du type `SOCK_STREAM`, cette fonction tente de se connecter à une autre *socket*. L'adresse de l'autre *socket* est indiquée par `addr`. Connect renvoie 0 s'il réussit, ou -1 s'il échoue.

5. Adresses du domaine `AF_INET` :

```
/* Adresse internet d'une machine */
struct in_addr {
        u_long  s_addr; /* codee en binaire */
}

/* Adresse internet d'une socket */
struct sockaddr_in {
        sa_family_t    sin_family;  /* AF_INET */
        in_port_t      sin_port;    /* numero du port associe */
        struct in_addr sin_addr;    /* adresse internet de la machine */
        char           sin_zero[8]; /*un champ de 8 caractères nuls */
};
```

   Si `sin_addr.s_addr` a comme valeur `INADDR_ANY`, la *socket* est associée à toutes les adresses de la machine (si celle-ci en possède plusieurs). Pour les machines possédant une seule adresse, la *socket* est associée à celle-ci (ce qui évite l'appel à la fonction `gethostname()`).

6. Convertir l'adresse Internet de l'hôte `in` en une chaîne de caractères dans la notation avec nombres et points :

```
# include <arpa/inet.h>
char * inet_ntoa (struct in_addr in);
```

   La chaîne est renvoyée dans un buffer alloué statiquement, qui est donc écrasé à chaque appel.

7. Conversion d'ordre des octets entre un hôte et un réseau. Sur les architectures i386, l'ordre des octets est différent de celui des réseaux comme Internet : l'octet de poids faible est placé en premier (on parle de LSB pour Least Significant Byte first), alors que sur les réseaux, c'est l'octet de poids fort en premier (on parle de MSB pour Most Significant Byte first). Il est donc nécessaire d'utiliser les fonctions suivantes pour effectuer correctement la conversion lors de la spécification des numéros de ports :

```
#include <netinet/in.h>
uint32_t htonl (uint32_t entierlong);  /* hôte -> réseau */
uint16_t htons (uint16_t entiercourt); /* hôte -> réseau */
uint32_t ntohl (uint32_t netlong);     /* réseau -> hôte */
uint16_t ntohs (uint16_t netshort);    /* réseau -> hôte */
```

8. Lecture / écriture (envoie) : avec `read()` et `write()`.