Institut Galilée — Architecture, systèmes, réseaux
Année 2005-2006 — L2

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`*.*

Correction 1. Tout d'abord le programme :

```
1  /* Le code client s'appelle ClientTCP */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <error.h>
6  #include <string.h>     /* memcpy(), strcmp() */
```

```
7   #include <unistd.h>
8   #include <netdb.h>       /* gethostbyname() */
9   #include <sys/types.h>
10  #include <sys/socket.h> /* socket(), connect(), bind(), ... */
11  #include <netinet/in.h> /* pour les sokets AF_INET, htonl(), etc. */
12  #include <arpa/inet.h>  /* inet_ntoa() */
13
14
15  #define ERROR(s) {perror(s);exit(-1);}
16  #define MAXCHAR  (1024)
17  #define SERVNAME (argv[1])
18  #define SERVPORT (atoi(argv[2]))
19
20  /*-------------------------------------------------------------------------*
21   *                                          routine d'acces au service   *
22   *-------------------------------------------------------------------------*/
23  void service(int sock){
24
25     while (1) {
26       char buffer[MAXCHAR];
27       int  i;
28
29       memset(buffer, '\0', MAXCHAR);
30
31       write(STDOUT_FILENO, "   vous> ", 9);
32
33       // scanf("%s", buffer); /* <---------<< s'arrete au premier espace */
34       // scanf("%[a-zA-Z .,;:_(')?!]", buffer); /* <----<< alternative 1 */
35       fgets(buffer, MAXCHAR, stdin); /* <----<< alternative 2 : recupere *
36                                       une ligne entiere, terminee par \n\0 */
37
38       if ( strcmp(buffer, "quit\n") == 0 ) {
39         break; /* <-------------------------------<< sortie de la boucle */
40       }
41
42       write (sock, buffer, strlen(buffer) + 1);   /* avec le \0 terminal */
43       i = read (sock, buffer, MAXCHAR);
44       write(STDOUT_FILENO, "serveur> ", 9);
45       write(STDOUT_FILENO, buffer, i);
46    }
47  }
48
49
50  /*-------------------------------------------------------------------------*
51   *                                                  programme principal    *
52   *-------------------------------------------------------------------------*/
53
54  int main(int argc, char *argv[]){
55    int                sock;
56    struct sockaddr_in adresseServeur;
57    struct hostent     *hp;
58
59    /* le nom du serveur est donne' en argument a la fonction main */
60    if ( argc != 3 ) {
61      fprintf(stderr, "Utilisation : %s nomServeur numeroPort\n",argv[0]);
62      exit(-1);
63    }
64
```

```
65    /* recuperer l'adresse du serveur */
66    hp = gethostbyname(SERVNAME);
67    if ( hp == NULL ) {
68      fprintf(stderr, "machine %s inconnue\n", argv[1]);
69      exit(-1);
70    }
71
72    /* creer une socket pour se connecter au serveur */
73    sock = socket(AF_INET, SOCK_STREAM, 0);
74    if ( sock == -1 ) {
75      ERROR ("Creation socket client");
76    }
77
78    adresseServeur.sin_family = AF_INET;
79    adresseServeur.sin_port = htons(SERVPORT);/* host to network short */
80    memcpy(&adresseServeur.sin_addr.s_addr, hp->h_addr, hp->h_length);
81
82    printf("Adresse du serveur : %s\n",
83               inet_ntoa(adresseServeur.sin_addr));/* net to ascii */
84
85    /* se connecter au serveur */
86    if ( -1 == connect(sock,
87                       (struct sockaddr *)&adresseServeur,
88                       sizeof(adresseServeur)) ) {
89      ERROR ("Connect");
90    }
91    printf("Connexion acceptee\n");
92
93    /* communiquer avec le serveur */
94    service(sock);
95
96    /* the end */
97    close(sock);
98
99    return 0;
100 }
```

1. **Le client demande la connexion avec `connect()`.**
2. **Des *sockets*. Il faut qu'il y ait une *socket* en attente sur le serveur et une *socket* à connecter sur le client.**
3. **Le client et le serveur dialoguent avec `read()` et `write()`.**
4. **`close(sock)` où `sock` est la socket met fin à la session.**
5. **Il faut convertir le nom du serveur en une adresse. Ça se fait avec `gethostbyname()` qui rend (une référence sur) un `struct hostent` dans lequel on aura l'adresse et sa longueur.**
6. **Les étapes importantes :** `gethostbyname()` ; `socket()` ; `connect()` ; $\{$`write()`, `read()`$\}^*$; `close()`.
7. **On passe la socket (un int) à la fonction de dialogue.**
8. **Voir le code.**
9. **Voir le code.**

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).*

Correction 2. C'est dans la plaquette du cours (schéma).

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 ?*

Correction 3.

1. **la première commande place un serveur TCP, sur le port 7777 de la machine, prêt à forker à la connexion avec un client, qui enverra au client ce qui est saisi sur stdin et affichera ce qu'il reçoit sur stdout (en bouclant indéfiniment). La seconde commande se connecte comme client TCP sur le port 7777 de la machine** `nom_de_machine` **(toujours en redirigeant la connexion vers stdin/stdin). La troisième commande lance un serveur qui, dès q'un client se connecte, fait exécuter par le shell la commande passée en premier argument (freud) et branche son entrée et sa sortie sur la socket. Freud est une version bash du psychiatre d'emacs, un descendant direct d'Eliza le premier chatbot (ne pas confondre avec tempest for Eliza, quand l'écran devient une carte son).**
2. **On peut connecter plusieurs clients au même serveur (parce qu'on fork) et la réponse tapée dans le terminal du serveur s'adressée (uniquement) au client qui a envoyé le dernier message.**
3. **Si on ne fork pas, un seul client peut se connecter à la fois. l'option** `-l` **est moins claire : la documentation dit qu'il s'agit d'accepter une nouvelle connexion après la connexion courante (c'est la réponse attendue). Dans ce cas cette option ne devrait pas être nécessaire en plus du fork. Mais, en réalité sans elle, et avec le fork, le dialogue avec un même client fonctionne mal.**

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 ?*

Correction 4.

1. **Avec un peu de chance quand on tape "psy", on est mis en communication avec freud.**
2. **Duplication de descripteurs de fichiers avec** `dup2()`**, de la socket vers l'entrée standard et vers la sortie standard puis appel à un** `exec...()`**. C'est sans doute une question trop difficile pour les étudiants.**

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()`.