

Rapport de presentation de projet  
Probleme 1 : LES SOCKETS

HOUNNOUKON Georges Cress  
VITOFODJI Jean Claude

SYSTEMES REPARTIS

# Contenu du Rapport

0.1	Enoncé . . . . .	3
0.2	Technologies et API utilisés . . . . .	4
0.2.1	Protocoles de communication . . . . .	4
0.2.2	Les Sockets . . . . .	4
0.3	Présentation des applications . . . . .	4
0.3.1	Application Serveur . . . . .	5
0.3.2	Application Client . . . . .	5
0.4	Captures d'écran . . . . .	6
0.5	Codes sources . . . . .	7
0.5.1	Serveur . . . . .	7
0.5.2	Client . . . . .	11

## Introduction

Conformément au programme de formation en Master de Systme d'Information et Réseaux Informatique, à l'institut de formation et de recherche en informatique de l'université d'Abomey-Calavi. Afin de comprendre et d'appliquer les notions reçues, il nous a été confié un TP sur l'utilisation des sockets en **C**<sup>1</sup> sous **Linux**<sup>2</sup>. Le projet consistera à réaliser une application client/serveur de transfert de fichiers<sup>3</sup>. Le présent rapport présente l'ensemble des travaux effectués et le mode d'utilisation des différentes applications.

---

1. Langage de programmation  
2. Système d'exploitation UNIX  
3. application client et serveur FTP

## 0.1 Enoncé

### Probleme sockets

Le but de cet exercice est de concevoir une application client/serveur de transfert de fichiers (sorte de ftpsimplifié). Vous devez donc écrire deux programmes : un client qui interagit avec l'utilisateur et un serveur qui interagit avec le client. De plus, client et serveur communiquent entre eux en utilisant les sockets comme interface de communication. En théorie, il est possible de choisir le type de protocole de communication (TCP ou UDP) ainsi que le mode de fonctionnement du serveur (itératif ou concurrent). Parmi ces 4 possibilités, vous ne mettrez en oeuvre que celle qui parat la plus réaliste.

### Description fonctionnelle de l'application

Soient **server** le **server** et **client** le client. En supposant que **server** s'exécute sur la machine `machine_du_serveur`, l'utilisateur travaillant sur la machine `machine_du_client` peut lancer la commande

— ***client machine\_du\_serveur***

pour démarrer une session de transfert de fichier. Les commandes possibles ainsi que le comportement souhaité de l'application sont alors :

- ***pwd*** : affiche le répertoire courant (de l'environnement) du serveur
- ***lpwd*** : affiche le répertoire courant (de l'environnement) du serveur
- ***cd repertoire*** : change le répertoire courant du serveur en repertoire
- ***lcd repertoire*** : change le répertoire courant du client en repertoire
- ***ls*** : affiche le contenu du répertoire courant du serveur
- ***lls*** : affiche le contenu du répertoire courant du client
- ***get fichier*** : “transfère” le fichier fichier du répertoire courant du serveur vers le répertoire courant du client
- ***put fichier*** : “transfère” le fichier fichier du répertoire courant du client vers le répertoire courant du serveur
- ***bye*** : termine la session de transfert de fichier

## 0.2 Technologies et API utilisés

### 0.2.1 Protocoles de communication

Le système à mettre en place devra permettre à deux application (Relativement différentes) à communiquer entre elles. Il est donc Important de mettre en place un protocole de communication. Et nous avons porté notre choix sur le **protocole TCP**. Ce choix s'explique par les possibilités qu'il nous offre et qui vont dans le mme sens que nos besoins. Il permet en effet de :

- faire des échanges en mode connecté
- assurer le transfert entier paquets entier de la source à la destination
- faire une conversation civilisée basée sur la demande et l'acceptation de connexion
- gérer une file d'attente pour le requetes en instance.

### 0.2.2 Les Sockets

Un socket représente une prise par laquelle une application peut envoyer et recevoir des donnes. Cette prise permet l'application de se brancher sur un rseau et communiquer avec d'autres applications qui y sont branches. Les informations crites sur une prise depuis une machine sont lues sur la prise d'une autre machine, et inversement. La fonction socket des API (application programming interface) sert crer un certain type de prise. Le type de prise sera choisi en fonction de la technologie de communication utiliser (par exemple TCP/IP). L'API permet un logiciel serveur de servir plusieurs clients simultanment. Une connexion est tablie entre le client et le serveur en vue de permettre la communication. La fonction connect permet un client de demander la connexion un serveur, et la fonction accept permet un serveur d'accepter cette connexion. Le programme serveur utilisera pralablement la fonction listen pour informer le logiciel sous-jacent qu'il est prt recevoir des connexions. Une fonction close permet de terminer la connexion. Lorsqu'un des deux interlocuteurs termine la connexion, l'autre est immdiatement avis. Une fois la connexion tablie, les fonctions send et recv servent respectivement envoyer et recevoir des informations.

## 0.3 Présentation des applications

Le fonctionnement du sytème est basé sur le principe de demande de *demande-traitement - réponse*. en effet, L'application serveur crée un socket qui se met en attente de connexion venant d'un client. L'application client de sont coté créé également un socket qui se connecte au socket créé par le serveur après acceptation de connexion. Le client et le serveur peuvent donc communiquer grace aux interfaces des sockets.

### 0.3.1 Application Serveur

Le serveur a pour rôle de démarrer et de rester en écoute en attente de connexion d'un client. dès qu'un client se connecte et envoie sa requête, ladite requête est exécutée au par le serveur et le résultat est renvoyé.

Pour démarrer le serveur, il faut lancer l'exécutable en précisant le port<sup>1</sup> de communication .

**exemple :** `./server 9090`

Ici le serveur démarre et se met en écoute sur le port 9090.

Il est à noter qu'aucune commande n'est saisie sur le serveur.

### 0.3.2 Application Client

L'application client se charge de se connecter au serveur et envoyer les requêtes au serveur. on la démarre en précisant en paramètre l'adresse ip du serveur et le numéro de port<sup>1</sup> de communication .

**exemple :** `./client 127.0.0.1 9090`

Ici le client démarre et se connecte au serveur à l'adresse 127.0.0.1 qui écoute le port 9090.

Une fois le client connecté au serveur on peut maintenant lancer les requêtes que sont :

- pwd
  - lpwd
  - cd repertoire
  - lcd repertoire
  - ls
  - ll
  - get fichier
  - put fichier
  - bye
- conformément au cahier des charges.

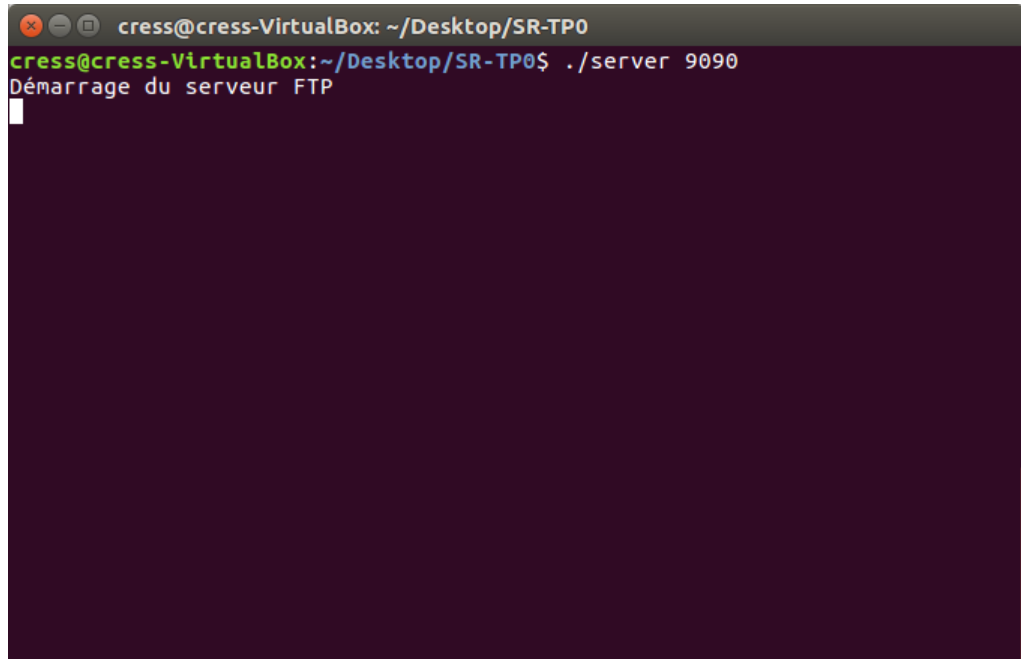
---

1. choisir un port non utilisé

1. celui choisi un port et utilisé sur le serveur

## 0.4 Captures d'écran

*Application serveur*

A screenshot of a terminal window titled 'cress@cress-VirtualBox: ~/Desktop/SR-TP0'. The terminal shows the command './server 9090' being executed, followed by the output 'Démarrage du serveur FTP' and a blank line with a cursor.

```
cress@cress-VirtualBox: ~/Desktop/SR-TP0
cress@cress-VirtualBox:~/Desktop/SR-TP0$ ./server 9090
Démarrage du serveur FTP

```

*Application client*

```
cress@cress-VirtualBox: ~/Desktop/SR-TP0
cress@cress-VirtualBox:~/Desktop/SR-TP0$ ./client 127.0.0.1 9090
127.0.0.1:9090~>/home/cress/Desktop/SR-TP0$ls
client
Files
fichier
fichier.c
fichier.c27
fichier.c2727
lettre.txt
Mes applications
Plage 2017
server
server.c
127.0.0.1:9090~>/home/cress/Desktop/SR-TP0$pwd
/home/cress/Desktop/SR-TP0
127.0.0.1:9090~>/home/cress/Desktop/SR-TP0$get fichier.c
Téléchargement en cours .Merci de patienter ...

Téléchargement terminé.

127.0.0.1:9090~>/home/cress/Desktop/SR-TP0$ls
client
client.c
fichier
fichier.c
fichier.c27
fichier.c2727
fichier.c272727
lettre.txt
Mes applications
```

## 0.5 Codes sources

### 0.5.1 Serveur

```
1 /*FTP server*/
2 #include <sys/socket.h>
3 #include <netinet/in.h>
4 #include <string.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 /*for getting file size using stat()*/
8 #include <sys/stat.h>
9 /*for sendfile()*/
10 #include <sys/sendfile.h>
11 /*for O_RDONLY*/
12 #include <fcntl.h>
13
14 int main(int argc, char *argv[])
15 {
16     printf("Demarrage du serveur FTP\n");
17     struct sockaddr_in server, client;
18     struct stat obj;
19     int sock1, sock2;
20     char buf[100], command[5], filename[20];
```



```

21  char * file;
22  file = malloc(100);
23  int k, i, size, len, c;
24  int filehandle;
25  sock1 = socket(AF_INET, SOCK_STREAM, 0);
26  printf("2\n");
27  if(sock1 == -1)
28  {
29      printf("Echec de creation du socket");
30      exit(1);
31  }
32  server.sin_port = htons(atoi(argv[1]));
33  server.sin_addr.s_addr = INADDR_ANY;
34  server.sin_family = AF_INET;
35  bzero(&(server.sin_zero), 8);
36  k = bind(sock1, (struct sockaddr*)&server, sizeof(server));
37  if(k == -1)
38  {
39      printf("Binding error\n");
40      exit(1);
41  }
42  k = listen(sock1, 1);
43  if(k == -1)
44  {
45      printf("Listen failed");
46      exit(1);
47  }
48  len = sizeof(struct sockaddr_in);
49  sock2 = accept(sock1, (struct sockaddr*)&client, &len);
50  printf("Nouvelle connexion > %d\n", sock2);
51  FILE *fp2 = popen("pwd", "r");
52  if (fgets(file, 100, fp2) != 0) {
53      send(sock2, file, strlen(file), 0);
54  }
55  i = 1;
56  while(1)
57  {
58      for (int i = 0; i < 100; ++i)
59      {
60          buf[i] = '\0';
61      }
62      recv(sock2, buf, 100, 0);
63      sscanf(buf, "%s", command);
64      if (!strcmp(command, "ls"))
65      {
66          FILE *fp2 = popen("ls", "r");
67          char buf[256];
68          int tail = 0;
69          while (fgets(file, 100, fp2) != 0) {
70              send(sock2, file, strlen(file), 0);
71              recv(sock2, buf, 100, 0);
72          }
73          send(sock2, "#end#", sizeof(char)*5, 0);
74      }
75      else if (!strcmp(command, "get"))
76      {
77          sscanf(buf, "%s%s", filename, filename);

```

```

78         stat(filename, &obj);
79         filehandle = open(filename, O_RDONLY);
80         size = obj.st_size;
81         if(filehandle == -1)
82             size = 0;
83         send(sock2, &size, sizeof(int), 0);
84         if(size)
85             sendfile(sock2, filehandle, NULL, size);
86
87     }
88     else if(!strcmp(command, "put"))
89     {
90         int c = 0, len;
91         char *f;
92         sscanf(buf+strlen(command), "%s", filename);
93         recv(sock2, &size, sizeof(int), 0);
94         i = 1;
95         while(1)
96         {
97             filehandle = open(filename, O_CREAT | O_EXCL | O_WRONLY, 0666);
98             if(filehandle == -1)
99             {
100                 sprintf(filename + strlen(filename), "%d", i);
101             }
102             else
103                 break;
104         }
105         f = malloc(size);
106         recv(sock2, f, size, 0);
107         c = write(filehandle, f, size);
108         close(filehandle);
109         send(sock2, &c, sizeof(int), 0);
110     }
111     else if(!strcmp(command, "pwd"))
112     {
113         FILE *fp2 = popen("pwd", "r");
114         if (fgets(file, 100, fp2) != 0) {
115             send(sock2, file, strlen(file), 0);
116         }
117     }
118     else if(!strcmp(command, "cd"))
119     {
120         if(chdir(buf+3) == 0){
121             FILE *fp2 = popen("pwd", "r");
122
123             if (fgets(file, 100, fp2) != 0) {
124                 send(sock2, file, strlen(file), 0);
125             }else{
126                 send(sock2, "#error", sizeof(char)*6, 0);
127             }
128         }else{
129             send(sock2, "#error", sizeof(char)*6, 0);
130         }
131     }
132     else if(!strcmp(command, "bye") || !strcmp(command, "quit"))
133     {
134         printf("FTP_server_quitting..\n");

```

```
135         i = 1;
136         send(sock2, &i, sizeof(int), 0);
137         exit(0);
138     }
139 }
140 return 0;
141 }
```

## 0.5.2 Client

```
1  /*FTP Client*/
2  #include <sys/socket.h>
3  #include <netinet/in.h>
4  #include <string.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  /*for getting file size using stat()*/
8  #include <sys/stat.h>
9  /*for sendfile()*/
10 #include <sys/sendfile.h>
11 /*for ORDONLY*/
12 #include <fcntl.h>
13 int main(int argc, char *argv[])
14 {
15     struct sockaddr_in server;
16     struct stat obj;
17     struct hostent *hostinfo = NULL;
18     //char *hostname ;
19     int sock;
20     int choice;
21     char buf[100], command[5], filename[20], *f;
22     int k, size, status;
23     int filehandle;
24     sock = socket(AF_INET, SOCK_STREAM, 0);
25     if(sock == -1)
26     {
27         printf("socket_creation_failed");
28         exit(1);
29     }
30
31     hostinfo = gethostbyname(argv[1]); /* on recupere les informations de l'hote auquel on veut
32     if (hostinfo == NULL) /* l'hote n'existe pas */
33     {
34         fprintf(stderr, "Unknown_host_%s.\n", argv[1]);
35         exit(EXIT_FAILURE);
36     }
37     for (int i = 0; i < 100; ++i)
38     {
39         buf[i] = '\0';
40     }
41     server.sin_family = AF_INET;
42     server.sin_port = htons(atoi(argv[2]));
43     server.sin_addr.s_addr = inet_addr(argv[1]);
44     k = connect(sock, (struct sockaddr*)&server, sizeof(server));
45     if(k == -1)
46     {
47         printf("Connect_Error");
48         exit(1);
49     }
50     int i = 1;
51     recv(sock, buf, 100, 0);
52     printf("\033[1;37m\tRepertoire->%s", buf);
53     while(1)
54     {
55         printf("\033[1;37mEnter_a_choice:\n1-get\n2-put\n3-pwd\n4-ls\n5-cd\n6-quit\n\033[0m");
```

```

56     scanf("%d", &choice);
57     for (int i = 0; i < 100; ++i)
58     {
59         buf[i] = '\0';
60     }
61     switch(choice)
62     {
63     case 1:
64         printf("Enter filename to get: ");
65         scanf("%s", filename);
66         strcpy(buf, "get ");
67         strcat(buf, filename);
68         send(sock, buf, 100, 0);
69         recv(sock, &size, sizeof(int), 0);
70         if(!size)
71         {
72             printf("No such file on the remote directory\n\n");
73             break;
74         }
75         f = malloc(size);
76         printf("T l chargement en cour...\n\n");
77         recv(sock, f, size, 0);
78         while(1)
79         {
80             filehandle = open(filename, O_CREAT | O_EXCL | O_WRONLY, 0666);
81             if(filehandle == -1)
82             {
83                 sprintf(filename + strlen(filename), "%d", i); //needed only
84             }
85             else break;
86         }
87         write(filehandle, f, size, 0);
88         close(filehandle);
89         printf("T l chargement termin .\n\n");
90         //strcpy(buf, "cat ");
91         //strcat(buf, filename);
92         //system(buf);
93         break;
94     case 2:
95         printf("Enter filename to put to server: ");
96         scanf("%s", filename);
97         filehandle = open(filename, O_RDONLY);
98         if(filehandle == -1)
99         {
100             printf("No such file on the local directory\n\n");
101             break;
102         }
103         strcpy(buf, "put ");
104         strcat(buf, filename);
105         send(sock, buf, 100, 0);
106         stat(filename, &obj);
107         size = obj.st_size;
108         send(sock, &size, sizeof(int), 0);
109         sendfile(sock, filehandle, NULL, size);
110         recv(sock, &status, sizeof(int), 0);
111         if(status)
112             printf("File stored successfully\n");

```

```

113         else
114             printf(" File _failed _to _be _stored _to _remote _machine\n");
115         break;
116     case 3:
117         strcpy(buf, "pwd");
118         send(sock, buf, 100, 0);
119         recv(sock, buf, 100, 0);
120         printf(" Repertoire _actuel : _\x1B[36m%s \033[0m", buf);
121         break;
122     case 4:
123         strcpy(buf, "ls");
124         send(sock, buf, 100, 0);
125         printf("\033[1;37m\tls _>\n");
126         printf("\x1B[36m");
127         while(1){
128             size = recv(sock, buf, 100, 0);
129             buf[size]='\0';
130
131             if(!strcmp(buf, "#end#")){
132                 break;
133             }
134             //buf[size-1]='\0';
135             printf("\t\t%s", buf);
136             send(sock, "ls", 100, 0);
137
138         }
139         printf("\033[0m");
140         break;
141     case 5:
142         strcpy(buf, "cd");
143         printf("\033[1;37m\tEntrer _l 'adresse _du _repertoire : _\033[0m");
144         scanf("%s", buf + 3);
145         send(sock, buf, 100, 0);
146         recv(sock, buf, 100, 0);
147         if(strcmp(buf, "#error"))
148             printf(" Repertoire _chang _> _\x1B[36m%s \033[0m", buf);
149         else
150             printf(" Impossible _de _changer _au _repertoire _demand !\n");
151         break;
152     case 6:
153         strcpy(buf, "quit");
154         send(sock, buf, 100, 0);
155         recv(sock, &status, 100, 0);
156         if(status)
157             {
158                 printf(" Server _closed \n Quitting ..\n");
159                 exit(0);
160             }
161         printf(" Server _failed _to _close _connection\n");
162         break;
163     }
164 }
165 }

```

## Conclusion

Le projet de réalisation d'application FTP a été pour nous une occasion de nous familiariser avec l'utilisation des sockets et surtout de comprendre comment programmer et déployer un système repartitionné. Nous avons beaucoup appris sur le langage C