

# Systemes et Réseaux : Implémentation d'un pseudo serveur FTP

Grand Maxence G1, Muller Lucie

06/04/2017

# Table des matières

# Introduction

# Chapitre 1

## Serveur FTP concurrent

### 1.1 Objectif de la réalisation

### 1.2 Code source commenté

#### 1.2.1 client.c

```
1 int main(int argc, char **argv)
2 {
3     int clientfd, fd;
4     char *buf, *host, *hidefile;
5     struct timeval start, end;
6     size_t len=0, n=0;
7
8     if (argc != 2) {
9         fprintf(stderr, "usage: %s <host>\n", argv[0]);
10        exit(0);
11    }
12    host = argv[1]; //On recupere le hostname du serveur (127.0.0.1 pour les tests)
13
14    clientfd = Open_clientfd(host, port); //On demande la connexion au serveur
15
16    req = malloc(sizeof(struct Request)); //On alloue de la memoire pour la Request
17    req->clientfd = clientfd; //On recupere le descripteur du socket
18
19    Rio_readinitb(&rio, clientfd); //On initialise Rio
20
21    printf("client connected to server %s\n", host);
22
23    while (1) {
24        printf("Tapez votre requete : \n");
25        buf = (char*)malloc(sizeof(char)*MAXLINE);
26        Fgets(buf, MAXLINE, stdin);
27        req->filename = (char*)malloc(sizeof(char)*MAXLINE);
28        req->cmd = (char*)malloc(sizeof(char)*MAXLINE);
29        req->content = (char*)malloc(sizeof(char)*MAXLINE);
30        sscanf(buf, "%s %s %s", req->cmd, req->filename, req->content); //On recupere la requete tapée
31        //par le client
32
33        if (!strcmp(req->cmd, "get")) {
34            gettimeofday(&start, NULL);
35            Rio_writen(req->clientfd, buf, strlen(buf)); //On envoie la requete au serveur
36            fd = open(strcat(req->filename, ".1"), O_RDWR | O_CREAT, 0666); //On creer un fichier pour le
37            //téléchargement
38            free(buf);
39            buf = (char*)malloc(sizeof(char)*MAXSEND);
40            while ((n=Rio_readn(req->clientfd, buf, MAXSEND)) > 0) { //Tant qu'on recoit des données, on
41            //les ecrit dans le fichier
42                rio_writen(fd, buf, n);
43                len+=n;
44                free(buf);
45                buf = (char*) malloc(sizeof(char)*MAXSEND);
46            }
47            Close(fd);
48            gettimeofday(&end, NULL);
```

```

double temps = ((end.tv_sec+(double)end.tv_usec/1000000)-(start.tv_sec+(double)start.tv_usec
/1000000));
47 printf("%lu bytes received in %f sec (%f bytes / sec) \n",len , temps , ((double)(len/temps)));
//On affiche le statistiques
49 if(len == 0){
    printf("Le tÃ©lÃ©chargement a echouÃ©\n");
    remove(req->filename);
51 }
} else if(!strcmp(req->cmd, "bye")){//On ferme la connexion
53 Rio_writen(req->clientfd, req->cmd, strlen(req->cmd));
    printf("fin de la connexion\n");
    exit(0);
55 } else{//Si la commande est inconnue on ferme la connexion
57 printf("Commande %s inconnue\n", req->cmd);
    printf("fin de la connexion\n");
    exit(0);
59 }
61 }
63 }

```

## 1.2.2 server.c

```

int main(int argc , char **argv)
2 {
    Signal(SIGINT, stop);
    Signal(SIGCHLD, handler);
    pid_t p;
    int listenfd , connfd;
    socklen_t clientlen;
    struct sockaddr_in clientaddr;
    char client_ip_string[INET_ADDRSTRLEN];
    char client_hostname[MAX_NAME_LEN];
10
    clientlen = (socklen_t)sizeof(clientaddr);

    listenfd = Open_listenfd(port);
    for(int i=0; i<NB_PROC; i++){
16         if((p=Fork())==0){
            while (1) {
18                 if((connfd = Accept(listenfd , (SA *)&clientaddr , &clientlen))>=0){
                    //On accepte la connexion entrante

                    /* determine the name of the client */
22                 Getnameinfo((SA *) &clientaddr , clientlen ,client_hostname , MAX_NAME_LEN,
0, 0, 0);

                    /* determine the textual representation of the client's IP address */
24                 Inet_ntop(AF_INET, &clientaddr.sin_addr , client_ip_string ,INET_ADDRSTRLEN)

;

26
                printf("server connected to %s (%s) %d\n", client_hostname ,
client_ip_string , getpid());
                req = malloc(sizeof(struct Request));
                req->connfd = connfd;
30
                readRequest(req);//On lit la requete du client
                Close(connfd);
32
            }
        }
        freeRequest(req);
        exit(0);
34
    }
    child[i] = p;
36
}
40
for(int i=0; i<NB_PROC; i++)
    waitpid(child[i] , NULL, 0);
42
//On attend tous les fils
44

```

```

46     exit(0);
47 }
48
49 void handler(int sig){
50     //On s'occupe des zombies
51     waitpid(-1, NULL, WNOHANG|WUNTRACED);
52     return;
53 }
54 void stop(int sig){
55     //Si on recoit un SIGCTR, on tue tous les fils
56     for(int i=0; i<NB_PROC; i++){
57         kill(SIGKILL, child[i]);
58     }
59     exit(0);
60 }
61
62 void readRequest(struct Request *req){
63
64     size_t n;
65     req->cmd = (char*)malloc(sizeof(char)*MAXLINE);
66     req->filename = (char*)malloc(sizeof(char)*MAXLINE);
67     req->content = (char*)malloc(sizeof(char)*MAXLINE);
68     char * request = (char *)malloc(sizeof(char)*MAXLINE);
69     rio_t rio;
70     Rio_readinitb(&rio, req->connfd);
71     n=Rio_readlineb(&rio, request, MAXLINE);
72     fflush(stdout);
73     request[n-1] = '\0';
74
75     sscanf(request, "%s %s %s", req->cmd, req->filename, req->content);
76     //On recupere la requete, et remplis notre Request
77     stat(req->filename, &req->sbuf);
78     if(!strcmp("get", req->cmd)){
79         get(req);
80     } else if (!strcmp("bye", req->cmd) ) {
81         printf("fin de la connexion");
82         Close(req->connfd);
83     }
84     fflush(stdout);
85 }
86
87 void get(struct Request *req){
88     char buf[MAXSEND];
89     size_t n, err;
90     rio_t rio;
91
92     //On envoie d'un coup
93     int srcfd;
94     char *srcp;
95     //Rio_readinitb(&rio, fd);
96     srcfd = Open(req->filename, O_RDONLY, 0);
97     srcp = Mmap(0, req->sbuf.st_size, PROT_READ, MAP_PRIVATE, srcfd, 0);
98     Close(srcfd);
99     Rio_writen(req->connfd, srcp, req->sbuf.st_size);
100     Munmap(srcp, req->sbuf.st_size);
101 }
102

```

## 1.3 Tests

```

maxence@Sybil:~$ cat Makefile
2 CC=gcc
3 CFLAGS=-Wall
4 LIBS=-lpthread
5
6 all: server client
7
8
9 server: server.o csapp.o tell_wait.o

```

```

10 $(CC) $(CFLAGS) -o server server.o csapp.o tell_wait.o $(LIBS)
12 client: client.o csapp.o
13 $(CC) $(CFLAGS) -o client client.o csapp.o $(LIBS)
14
15 client.o: client.c csapp.c client.h csapp.h
16 $(CC) $(CFLAGS) -c client.c csapp.c $(LIBS)
18 server.o: server.c csapp.c server.h csapp.h
19 $(CC) $(CFLAGS) -c server.c csapp.c $(LIBS)
20
21 tell_wait.o: tell_wait.c csapp.c tell_wait.h csapp.h
22 $(CC) $(CFLAGS) -c tell_wait.c csapp.c $(LIBS)
24 csapp.o: csapp.c csapp.h
25 $(CC) $(CFLAGS) -c csapp.c $(LIBS)
26
27 clean:
28 rm server client *.o
maxence@Sybil:~$ ls
30 backupServer client.c client.o csapp.h Makefile server server.h tell_wait.c tell_wait.o
   test_pipe.c
   client client.h csapp.c csapp.o master.c server.c server.o tell_wait.h test
32 maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
34 Tapez votre requete :
get Makefile
36 617 bytes received in 0.002723 sec (226590.103143 bytes / sec)
Tapez votre requete :
38 bye
fin de la connexion
40 maxence@Sybil:~$ cat Makefile1
CC=gcc
42 CFLAGS=-Wall
LIBS=-lpthread
44
45 all: server client
46
47 server: server.o csapp.o tell_wait.o
48 $(CC) $(CFLAGS) -o server server.o csapp.o tell_wait.o $(LIBS)
50
51 client: client.o csapp.o
52 $(CC) $(CFLAGS) -o client client.o csapp.o $(LIBS)
54
55 client.o: client.c csapp.c client.h csapp.h
56 $(CC) $(CFLAGS) -c client.c csapp.c $(LIBS)
58
59 server.o: server.c csapp.c server.h csapp.h
60 $(CC) $(CFLAGS) -c server.c csapp.c $(LIBS)
62
63 tell_wait.o: tell_wait.c csapp.c tell_wait.h csapp.h
64 $(CC) $(CFLAGS) -c tell_wait.c csapp.c $(LIBS)
66
67 csapp.o: csapp.c csapp.h
68 $(CC) $(CFLAGS) -c csapp.c $(LIBS)
69
70 clean:
71 rm server client *.o

```

# Chapitre 2

## Découpage du fichier

### 2.1 Objectif de la réalisation

### 2.2 Code source commenté

#### 2.2.1 server.c

Seul la fonction get de server.c a été modifiée

```
1 void get(struct equest *req){
    char buf[MAXSEND];
3     size_t n, err;
    rio_t rio;

5     int fd = open(req->filename, O_RDONLY);
    Rio_readinitb(&rio, fd);
7     /*
    Tant qu'on a écrit, on continue d'écrire
    */
9     while ((n = Rio_readlineb(&rio, buf, MAXSEND)) > 0) {
        err = rio_writen(req->connfd, buf, n);
11        if(err == -1){/*On stopper si le client s'arrete
            fprintf(stderr, "Arret innatendu du client\n");
13            break;
        }
15    }
17 }
19 }
```

### 2.3 Tests

```
1 maxence@Sybil:~$ cat test
backupServer
3 client
client.c
5 client.h
client.o
7 csapp.c
csapp.h
9 csapp.o
Makefile
11 Makefile1
master.c
13 server
server.c
15 server.h
server.o
17 tell_wait.c
tell_wait.h
19 tell_wait.o
```



```
test
21 test_pipe.c
maxence@Sybil:~$ ./client 127.0.0.1
23 client connected to server 127.0.0.1
Tapez votre requete :
25 get test
186 bytes received in 0.009943 sec (18706.611932 bytes / sec)
27 Tapez votre requete :
bye
29 fin de la connexion
maxence@Sybil:~$ cat test1
31 backupServer
client
33 client.c
client.h
35 client.o
csapp.c
37 csapp.h
csapp.o
39 Makefile
Makefile1
41 master.c
server
43 server.c
server.h
45 server.o
tell_wait.c
47 tell_wait.h
tell_wait.o
49 test
test_pipe.c
```

## Chapitre 3

# Gestion simple des pannes côté client

### 3.1 Objectif de la réalisation

### 3.2 Code source commenté

Seul le fichier server.c a été modifié

#### 3.2.1 client.c

```
/*
2 Pour savoir si on a déjà tenté de télécharger un fichier, on créer un fichier temporaire au
  début du téléchargement, t
Lorsque le téléchargement est terminé on le supprime
4 Si il existe un fichier temporaire associé au fichier que lon veut télécharger
  cela signifie qu'on a déjà essayé de le télécharger, on télécharge uniquement les données
  manquantes
6 */
if(!strcmp(req->cmd, "get")){
8   gettimeofday(&start, NULL);
  Rio_writen(req->clientfd, buf, strlen(buf)); //On envoie la requete au serveur
10  hidefile = (char*) malloc(sizeof(char)*MAX_NAME_LEN);
  hidefile[0] = '\0';
12  strcat(hidefile, req->filename); //On créer le nom du fichier temporaire
  if(stat(hidefile, &req->sbuf) != 0){ //Si le fichier temporaire n'existe pas
14    open(hidefile, O_RDWR | O_CREAT, 0666); //On le créer
    fd = open(strcat(req->filename, ".1"), O_RDWR | O_CREAT, 0666); //On créer un fichier pour le
    téléchargement
16    free(buf);
    buf = (char*) malloc(sizeof(char)*MAXSEND);
18    while((n=Rio_readn(req->clientfd, buf, MAXSEND)) > 0) { //Tant qu'on recoit des données, on les
      écrit dans le fichier
        rio_writen(fd, buf, n);
20        len+=n;
        free(buf);
22        buf = (char*) malloc(sizeof(char)*MAXSEND);
    }
24    Close(fd);
    gettimeofday(&end, NULL);
26    double temps = ((end.tv_sec+(double)end.tv_usec/1000000)-(start.tv_sec+(double)start.tv_usec/1000000));
    printf("%lu bytes received in %f sec (%f bytes / sec) \n", len, temps, ((double)(len/temps))); //
    On affiche le statistiques
28    if(len == 0){
      printf("Le téléchargement a échoué\n");
30    }
    remove(req->filename);
  }
  remove(hidefile);
} else { //Si le fichier temporaire existe
34  fd = open(strcat(req->filename, ".1"), O_WRONLY);
  stat(req->filename, &req->sbuf);
36  int dejaLu = req->sbuf.st_size; //On recupere la taille des données déjà téléchargées
  free(buf);
38  buf = (char*) malloc(sizeof(char)*MAXSEND);
  while((n=Rio_readn(req->clientfd, buf, MAXSEND)) > 0) {
```

```

40     int tmp = dejaLu;
    dejaLu -= n;
42     if (dejaLu <= 0) { // On écrit si ce sont des données non téléchargées
        rio_writen(fd, buf, n);
44         free(buf);
        buf = (char*) malloc(sizeof(char)*MAXSEND);
46         len += n;
    } else { // Sinon on déplace le curseur
48         if (n < tmp)
            lseek(fd, n, SEEK_CUR);
50         else
            lseek(fd, n - tmp, SEEK_CUR);
52     }
    }
54     gettimeofday(&end, NULL);
    double temps = ((end.tv_sec + (double)end.tv_usec/1000000) - (start.tv_sec + (double)start.tv_usec/1000000));
56     printf("%lu bytes received in %f sec (%f bytes / sec) \n", len, temps, ((double)(len/temps)));
    remove(hidefile);
58 }
} else if (!strcmp(req->cmd, "bye")) { // On ferme la connexion
60     Rio_writen(req->clientfd, req->cmd, strlen(req->cmd));
    printf("fin de la connexion\n");
62     exit(0);
} else { // On stoppe la connexion
64     printf("Commande %s inconnue\n", req->cmd);
    printf("fin de la connexion\n");
66     exit(0);
}
68 }

```

### 3.3 Tests

```

maxence@Sybil:~$ rm test1
2 maxence@Sybil:~$ head -3 test > test1
maxence@Sybil:~$ touch Ttest
4 maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
6 Tapez votre requete :
get test
8 186 bytes received in 0.000816 sec (227977.949737 bytes / sec)
Tapez votre requete :
10 bye
fin de la connexion
12 maxence@Sybil:~$ cat test1
backupServer
14 client
client.c
16 client.h
client.o
18 csapp.c
csapp.h
20 csapp.o
Makefile
22 Makefile1
master.c
24 server
server.c
26 server.h
server.o
28 tell_wait.c
tell_wait.h
30 tell_wait.o
test

```

## Chapitre 4

# Serveur FTP avec équilibrage des charges

### 4.1 Objectif de la réalisation

### 4.2 Code source commenté

#### 4.2.1 master.c

```
1  int sCourant = 0;
3  int slave[NB_PROC];
5  int next(){//Renvoie le prochain esclave
    int p = slave[sCourant];
7   sCourant = ++sCourant % NB_PROC;
    return p;
9  }
11 int main(int argc, char **argv)
12 {
13     int listenfd, connfd;
14     socklen_t clientlen;
15     struct sockaddr_in clientaddr;
16     char client_ip_string[INET_ADDRSTRLEN];
17     char client_hostname[MAX_NAME_LEN];
18     rio_t rio;
19
20     clientlen = (socklen_t) sizeof(clientaddr);
21
22     listenfd = Open_listenfd(port);
23
24     for(int i = 0; i < NB_PROC; i++){
25         slave[i] = port + i + 1;
26     }
27     while(1){
28         if((connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen)) >= 0){
29
30             /* determine the name of the client */
31             Getnameinfo((SA *) &clientaddr, clientlen, client_hostname, MAX_NAME_LEN, 0, 0, 0);
32
33             /* determine the textual representation of the client's IP address */
34             Inet_ntop(AF_INET, &clientaddr.sin_addr, client_ip_string, INET_ADDRSTRLEN);
35
36             printf("server connected to %s (%s) %d\n", client_hostname, client_ip_string, getpid());
37
38             Rio_readinitb(&rio, connfd);
39             int p = next();
40             Rio_writen(connfd, &p, sizeof(p)); //Envoie au client le numero de port de l'esclave traitant
41
42             Close(connfd); //Ferme la connexion avec le client
43         }
44     }
45     exit(0);
46 }
```

### 4.2.2 server.c

```
1 clientfd = Open_clientfd(host, port); //On demande la connexion au serveur maitre
2 printf("client connected to server %s\n", host);
3
4 Rio_readn(clientfd, &port2, sizeof(int)); //On recupert le port du serveur esclave
5
6 printf("Vous aller Ãatre redirig  vers l'esclave %d\n", port2);
7
8 clientfd = Open_clientfd(host, port2); //On demande la connexion au serveur esclave
9
10 req = malloc(sizeof(struct Request)); //On alloue de la memoire pour la Request
11 req->clientfd = clientfd; //On recupere le descripteur du socket
12
```

## 4.3 Tests

```
1 maxence@Sybil:~$ ls > test
2 maxence@Sybil:~$ ./client 127.0.0.1
3 client connected to server 127.0.0.1
4 Vous aller Ãatre redirig  vers l'esclave 2122
5 Tapez votre requete :
6 get test
7 238 bytes received in 0.004584 sec (51921.582857 bytes / sec)
8 Tapez votre requete :
9 bye
10 fin de la connexion
11 maxence@Sybil:~$ cat test
12 backupServer
13 client
14 client.c
15 client.h
16 client.o
17 csapp.c
18 csapp.h
19 csapp.o
20 debug
21 Makefile
22 master
23 master.c
24 master.o
25 projetFTPBackup
26 rapport.pdf
27 rapport.tex
28 rapport.toc
29 server
30 server.c
31 server.h
32 server.o
33 tell_wait.c
34 tell_wait.h
35 tell_wait.o
36 test
```

## Chapitre 5

# Plusieurs demandes de fichiers par connexions

5.1 Objectif de la réalisation

5.2 Code source commenté

5.3 Tests

## Chapitre 6

# Commandes ps, pwd et cd

6.1 Objectif de la réalisation

6.2 Code source commenté

6.3 Tests

## Chapitre 7

# Commande mkdir, rm, rmm -r et put

7.1 Objectif de la réalisation

7.2 Code source commenté

7.3 Tests



## Chapitre 8

# Authentification

8.1 Objectif de la réalisation

8.2 Code source commenté

8.3 Tests