

Systemes et Réseaux : Implémentation d'un pseudo serveur FTP

Grand Maxence, Muller Lucie (Groupe 1)

09/04/2017

Table des matières

1	Serveur FTP concurrent	3
1.1	Objectif de la réalisation	3
1.2	Code source commenté	3
1.2.1	client.c	3
1.2.2	server.c	4
1.3	Tests	5
2	Découpage du fichier	7
2.1	Objectif de la réalisation	7
2.2	Code source commenté	7
2.2.1	server.c	7
2.3	Tests	7
3	Gestion simple des pannes coté client	9
3.1	Objectif de la réalisation	9
3.2	Code source commenté	9
3.2.1	client.c	9
3.3	Tests	10
4	Serveur FTP avec équilibrage des charges	11
4.1	Objectif de la réalisation	11
4.2	Code source commenté	11
4.2.1	master.c	11
4.2.2	server.c	12
4.3	Tests	12
5	Plusieurs demandes de fichiers par connexions	13
5.1	Objectif de la réalisation	13
5.2	Code source commenté	13
5.2.1	server.c	13
5.3	Tests	13
6	Commandes ls, pwd et cd	15
6.1	Objectif de la réalisation	15
6.2	Code source commenté	15
6.2.1	server.c	15
6.3	Tests	16
7	Authentification	17
7.1	Objectif de la réalisation	17
7.2	Code source commenté	17
7.2.1	master.c	17
7.3	Tests	17
8	Les commandes rm, rm -r, mkdir et put	19
8.1	code source commenté	19
8.1.1	server.c	19
8.1.2	client.c	20

Introduction

Durant ce projet, nous avons du implémenter un serveur de fichier, inspiré des serveurs FTP. Nous commencé avec un serveur doté d'une fonction simple : (la récupération de fichiers à distance), puis nous aons implanter un serveur permettant des actions plus complexes et, de ce fait, plus performant.

Pour ce faire nous avons créer une structure Request permettant de récuérer toutes les informations utiles :

```
1 struct Request{
2     char * cmd; //La methode
3     char * filename; //Le fichier a prendre en commpte
4     char * content; //informtion supplementaire
5     int connfd; //descripteur de socket
6     int fini; //1 si bye envoye, 0 sinon
7     struct stat sbuf;
8 };
9
```

Chapitre 1

Serveur FTP concurrent

1.1 Objectif de la réalisation

Dans cette partie, nous avons du implémenter la fonction simple du serveur : la récupération de fichier à distance en donnant le nom du fichier à partir du client. Le serveur crée un nouveau fichier sur lequel, à l'aide d'un buffer, il copie le contenu du premier fichier. Ce nouveau fichier est sauvegardé à l'emplacement courant du client, sous le nom "nomdupremier fichier1"

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 tapee
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            //tÃ©lÃ©chargement
38            free(buf);
39            buf = (char*)malloc(sizeof(char)*MAXSEND);
```

```

38     while((n=Rio_readn(req->clientfd, buf, MAXSEND)) > 0) { //Tant qu'on recoit des donnees, on les
ecrit dans le fichier
        rio_writen(fd, buf, n);
        len+=n;
        free(buf);
        buf = (char*) malloc(sizeof(char)*MAXSEND);
    }
    Close(fd);
    gettimeofday(&end, NULL);
    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 les statistiques
    if(len == 0){
        printf("Le tÃ©lÃ©chargement a echouÃ©\n");
        remove(req->filename);
    }
52 } else if(!strcmp(req->cmd, "bye")){ //On ferme la connexion
    Rio_writen(req->clientfd, req->cmd, strlen(req->cmd));
54     printf("fin de la connexion\n");
    exit(0);
56 } else { //Si la commande est inconnue on ferme la connexion
    printf("Commande %s inconnue\n", req->cmd);
    printf("fin de la connexion\n");
    exit(0);
60 }
62 }
}

```

1.2.2 server.c

```

    int main(int argc, char **argv)
    {
        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];

        clientlen = (socklen_t) sizeof(clientaddr);

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

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

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

;

                        printf("server connected to %s (%s) %d\n", client_hostname,
client_ip_string, getpid());
                        req = malloc(sizeof(struct Request));
                        req->connfd = connfd;

                        readRequest(req); //On lit la requete du client
                        Close(connfd);
                    }
                }
            }
            freeRequest(req);
        }
    }
}

```

```

38         exit(0);
39     }
40     child[i] = p;
41 }
42 for(int i=0; i<NB_PROC; i++)
43     waitpid(child[i], NULL, 0);
44 //On attend tous les fils
45
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
  CFLAGS=-Wall
4 LIBS=-lpthread
  all: server client
6 server: server.o csapp.o tell_wait.o
  $(CC) $(CFLAGS) -o server server.o csapp.o tell_wait.o $(LIBS)
8
  client: client.o csapp.o
10 $(CC) $(CFLAGS) -o client client.o csapp.o $(LIBS)
  client.o: client.c csapp.c client.h csapp.h
12 $(CC) $(CFLAGS) -c client.c csapp.c $(LIBS)

14 server.o: server.c csapp.c server.h csapp.h
  $(CC) $(CFLAGS) -c server.c csapp.c $(LIBS)
16 tell_wait.o: tell_wait.c csapp.c tell_wait.h csapp.h
  $(CC) $(CFLAGS) -c tell_wait.c csapp.c $(LIBS)
18
csapp.o: csapp.c csapp.h
20 $(CC) $(CFLAGS) -c csapp.c $(LIBS)
clean:
22 rm server client *.o
maxence@Sybil:~$ ls
24 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
26 maxence@Sybil:~$ ./client 127.0.0.1
  client connected to server 127.0.0.1
28 Tapez votre requete :
  get Makefile
30 617 bytes received in 0.002723 sec (226590.103143 bytes / sec)
  Tapez votre requete :
32 bye
  fin de la connexion
34 maxence@Sybil:~$ cat Makefile1
  CC=gcc
36 CFLAGS=-Wall
  LIBS=-lpthread
38
  all: server client
40
42 server: server.o csapp.o tell_wait.o
  $(CC) $(CFLAGS) -o server server.o csapp.o tell_wait.o $(LIBS)
44 client: client.o csapp.o
  $(CC) $(CFLAGS) -o client client.o csapp.o $(LIBS)
46
  client.o: client.c csapp.c client.h csapp.h
48 $(CC) $(CFLAGS) -c client.c csapp.c $(LIBS)
  server.o: server.c csapp.c server.h csapp.h
50 $(CC) $(CFLAGS) -c server.c csapp.c $(LIBS)

52 tell_wait.o: tell_wait.c csapp.c tell_wait.h csapp.h
  $(CC) $(CFLAGS) -c tell_wait.c csapp.c $(LIBS)
54 csapp.o: csapp.c csapp.h
  $(CC) $(CFLAGS) -c csapp.c $(LIBS)
56
clean:
58 rm server client *.o

```

Chapitre 2

Découpage du fichier

2.1 Objectif de la réalisation

Dans cette partie, nous avons dû améliorer le serveur afin de ne pas envoyer le fichier en une seule fois, mais en plusieurs paquets. Cela permet de ne pas surcharger la mémoire si un fichier de grande taille est transféré. Pour cela on utilise une boucle qui envoie à chaque fois MAXSEND (maximum d'octets envoyés) octets au client.

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){
2     char buf[MAXSEND];
3     size_t n, err;
4     rio_t rio;
5
6     int fd = open(req->filename, O_RDONLY);
7     Rio_readinitb(&rio, fd);
8     /*
9     Tant qu'on a écrit, on continue d'écrire
10    */
11    while ((n = Rio_readlineb(&rio, buf, MAXSEND)) > 0) {
12        err = rio_writen(req->connfd, buf, n);
13        if(err == -1){ // On stoppe si le client s'arrête
14            fprintf(stderr, "Arret innatendu du client\n");
15            break;
16        }
17    }
18 }
19
20 }
```

2.3 Tests

```
1 maxence@Sybil:~$ cat test
2 backupServer
3 client
4 client.c
5 client.h
6 client.o
7 csapp.c
8 csapp.h
9 csapp.o
10 Makefile
11 Makefile1
12 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 coté client

3.1 Objectif de la réalisation

Dans cette partie, nous devons gérer les pannes du client, c'est-à-dire le comportement du serveur au cas où le client s'arrêterait durant le transfert. Un fichier temporaire est créé au début du processus et lorsque ce dernier est terminé le fichier est supprimé. Il n'est pas supprimé si le client s'arrête avant la fin du téléchargement. Le client vérifie, à l'appel du transfert, si un tel fichier temporaire existe. Si c'est le cas cela signifie que le serveur a déjà commencé un transfert mais que celui-ci a été arrêté avant sa terminaison. Le client récupère alors ce fichier temporaire et sait quelle quantité de donnée il lui reste à télécharger.

3.2 Code source commenté

Seul le fichier server.c a été modifié

3.2.1 client.c

```
1 if(!strcmp(req->cmd, "get")){
2     gettimeofday(&start, NULL);
3     Rio_writen(req->clientfd, buf, strlen(buf)); //On envoie la requete au serveur
4     hidefile = (char*)malloc(sizeof(char)*MAX_NAME_LEN);
5     hidefile[0] = '\0';
6     strcat(hidefile, req->filename); //On cree le nom du fichier temporaire
7     if(stat(hidefile, &req->sbuf) != 0) { //Si le fichier temporaire n'existe pas
8         open(hidefile, O_RDWR | O_CREAT, 0666); //On le cree
9         fd = open(strcat(req->filename, ".tmp"), O_RDWR | O_CREAT, 0666); //On cree un fichier pour le
10            tÃ©lÃ©chargement
11        free(buf);
12        buf = (char*)malloc(sizeof(char)*MAXSEND);
13        while((n=Rio_readn(req->clientfd, buf, MAXSEND)) > 0) { //Tant qu'on recoit des donnees, on les
14            ecrit dans le fichier
15            rio_writen(fd, buf, n);
16            len+=n;
17            free(buf);
18            buf = (char*) malloc(sizeof(char)*MAXSEND);
19        }
20        Close(fd);
21        gettimeofday(&end, NULL);
22        double temps = ((end.tv_sec+(double)end.tv_usec/1000000)-(start.tv_sec+(double)start.tv_usec/1000000));
23        printf("%lu bytes received in %f sec (%f bytes / sec) \n", len, temps, ((double)(len/temps))); //
24        On affiche les statistiques
25        if(len == 0){
26            printf("Le tÃ©lÃ©chargement a echoue\n");
27            remove(req->filename);
28        }
29        remove(hidefile);
30    } else { //Si le fichier temporaire existe
31        fd = open(strcat(req->filename, ".tmp"), O_WRONLY);
32        stat(req->filename, &req->sbuf);
33        int dejaLu = req->sbuf.st_size; //On recupere la taille des donnees deja telechargees
34        free(buf);
```

```

32 buf = (char*) malloc(sizeof(char)*MAXSEND);
while((n=Rio_readn(req->clientfd, buf, MAXSEND)) > 0) {
34     int tmp = dejaLu;
    dejaLu=-n;
36     if(dejaLu <= 0){ //On ecrit si ce sont des donnees non tÃ©lÃ©chargÃ©es
        rio_writen(fd, buf, n);
38         free(buf);
        buf = (char*) malloc(sizeof(char)*MAXSEND);
40         len+=n;
    } else { //Sinon on dÃ©place le curseur
42         if(n<tmp)
            lseek(fd, n, SEEK_CUR);
44         else
            lseek(fd, n-tmp, SEEK_CUR);
46     }
    }
48     gettimeofday(&end, NULL);
    double temps = ((end.tv_sec+(double)end.tv_usec/1000000)-(start.tv_sec+(double)start.tv_usec
/1000000));
50     printf("%lu bytes received in %f sec (%f bytes / sec) \n",len, temps, ((double)(len/temps)));
    remove(hidefile);
52 }
} else if(!strcmp(req->cmd, "bye")){ //On ferme la connexion
54     Rio_writen(req->clientfd, req->cmd, strlen(req->cmd));
    printf("fin de la connexion\n");
56     exit(0);
} else { //On stop la connexion
58     printf("Commande %s inconnue\n", req->cmd);
    printf("fin de la connexion\n");
60     exit(0);
}
62 }

```

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

Dans cette partie, nous avons ajouté un répartisseur de charge. Il sert à partager les différentes requêtes des clients entre plusieurs serveurs dit "esclaves". Cette répartition est gérée par un serveur "maître" qui reçoit toutes les requêtes avant de les envoyer aux esclaves. Lorsque le serveur maître reçoit une connexion venant d'un client, il lui renvoie le port d'un esclave, auquel le client doit se connecter.

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

```

43     }
44     }
45     exit(0);
46 }
47

```

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 redirigÃ© 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 redirigÃ© 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

Dans cette partie, nous avons modifié le serveur afin qu'il gère différentes requêtes d'un client sans fermer la connexion entre chacun d'elle. En d'autres termes il peut ainsi les gérer les unes après les autres sans avoir à se reconnecter. La fermeture de connexion se fait via le client avec une commande "bye".

5.2 Code source commenté

5.2.1 server.c

Tout d'aboord on rejoute l'option suivante dans la fonction readRequest :

```
1 } else if (!strcmp("bye", req->cmd) ) { //Si le client envoie bye, on ferme la connexion
2     printf("fin de la connexion\n");
3     Close(req->connfd);
4     req->fini=1;
5 }
6
```

Et pour le main, maintenant on lit les requêtes envoyées tant que l'on a pas reçue bye :

```
1 while(!req->fini){
2     readRequest(req, 0); //On lit la requete du client
3 }
```

5.3 Tests

```
1 maxence@Sybil:~$ ls > test
2 maxence@Sybil:~$ ls -l > testt
3 maxence@Sybil:~$ ./client 127.0.0.1
4 client connected to server 127.0.0.1
5 Vous allez Ãatre redirigÃ© vers l'esclave 2122
6 Tapez votre requete :
7 get test
8 262 bytes received in 5.005285 sec (52.344671 bytes / sec)
9 Tapez votre requete :
10 get testt
11 1969 bytes received in 5.008208 sec (393.154595 bytes / sec)
12 Tapez votre requete :
13 bye
14 fin de la connexion
15 maxence@Sybil:~$ cat test1
16 backupServer
```

```

17 client
   client.c
19 client.cl
   client.h
21 client.o
   csapp.c
23 csapp.h
   csapp.o
25 debug
   Makefile
27 master
   master.c
29 master.o
   projetFTPBackup
31 rapport.aux
   rapport.log
33 rapport.pdf
   rapport.tex
35 rapport.toc
   server
37 server.c
   server.h
39 server.o
   t
41 tell_wait.c
   tell_wait.h
43 test
maxence@Sybil:~$ cat testt1
45 total 3456
-rwx----- 1 maxence maxence 3294 avril 1 11:41 backupServer
47 -rwx----- 1 maxence maxence 36488 avril 9 18:29 client
-rwx----- 1 maxence maxence 7025 avril 9 18:21 client.c
49 -rwx----- 1 maxence maxence 7018 avril 9 16:23 client.cl
-rwx----- 1 maxence maxence 224 avril 4 16:05 client.h
51 -rwx----- 1 maxence maxence 9952 avril 9 18:21 client.o
-rwx----- 1 maxence maxence 20259 mars 29 18:33 csapp.c
53 -rwx----- 1 maxence maxence 6105 mars 29 18:33 csapp.h
-rwx----- 1 maxence maxence 23760 avril 9 18:29 csapp.o
55 -rwx----- 1 maxence maxence 0 avril 9 16:45 debug
-rwx----- 1 maxence maxence 960 avril 9 16:24 Makefile
57 -rwx----- 1 maxence maxence 36704 avril 9 18:29 master
-rwx----- 1 maxence maxence 6867 avril 9 18:27 master.c
59 -rwx----- 1 maxence maxence 10136 avril 9 18:29 master.o
drwx----- 1 maxence maxence 131072 avril 9 18:14 projetFTPBackup
61 -rwx----- 1 maxence maxence 4046 avril 9 18:39 rapport.aux
-rwx----- 1 maxence maxence 44067 avril 9 18:39 rapport.log
63 -rwx----- 1 maxence maxence 259857 avril 9 18:39 rapport.pdf
-rwx----- 1 maxence maxence 22400 avril 9 18:39 rapport.tex
65 -rwx----- 1 maxence maxence 2538 avril 9 18:39 rapport.toc
-rwx----- 1 maxence maxence 36776 avril 9 18:29 server
67 -rwx----- 1 maxence maxence 6491 avril 9 18:28 server.c
-rwx----- 1 maxence maxence 452 avril 9 16:04 server.h
69 -rwx----- 1 maxence maxence 10728 avril 9 18:29 server.o
-rwx----- 1 maxence maxence 1794 fÃ©vr. 28 09:24 tell_wait.c
71 -rwx----- 1 maxence maxence 131 fÃ©vr. 28 08:55 tell_wait.h
-rwx----- 1 maxence maxence 262 avril 9 18:42 test
73 -rwx----- 1 maxence maxence 0 avril 9 18:42 testt

```

Chapitre 6

Commandes ls, pwd et cd

6.1 Objectif de la réalisation

Dans cette partie, nous avons ajouté les commandes ls (permettant d’afficher le contenu du répertoire courant du serveur FTP), pwd (permettant d’afficher le chemin courant) et cd (permettant de changer le répertoire courant du serveur FTP). Pour les commandes ls et pwd, le serveur crée un processus fils qui exécute la commande à l’aide de `execvp`, et pour cd, nous utilisons la fonction `chdir`.

6.2 Code source commenté

6.2.1 server.c

```
1  if (!strcmp("ls", req->cmd)){
2      char* args[1];
3  args[0] = req->cmd;
4  int c;
5  int fd = open(".log", O_RDWR | O_CREAT, 0666);
6  if ((c=Fork())==0){
7      close(1);
8      dup(fd);
9      execvp(args[0], args);
10     exit(0);
11 }
12 close(fd);
13 waitpid(c, NULL, 0);
14 memcpy(req->filename, ".log", 4);
15 stat(req->filename, &req->sbuf);
16 int l = req->sbuf.st_size;
17 rio_writen(req->connfd, &l, sizeof(int));
18 get(req);
19 remove(".log");
20 } else if (!strcmp("pwd", req->cmd)){
21     char* args[1];
22     args[0] = "pwd";
23     int c;
24     int fd = open(".log", O_RDWR | O_CREAT, 0666);
25     if ((c=Fork())==0){
26         close(1);
27         dup(fd);
28         execvp(args[0], args);
29         exit(0);
30     }
31     close(fd);
32     waitpid(c, NULL, 0);
33     memcpy(req->filename, ".log", 4);
34     stat(req->filename, &req->sbuf);
35     int l = req->sbuf.st_size;
36     rio_writen(req->connfd, &l, sizeof(int));
37     get(req);
38     remove(".log");
39 } else if (!strcmp("cd", req->cmd)){
```



```
41     chdir(req->filename); //On se deplace dans req->filename (qui est ici un repertoire et non pas un
    fichier)
}
```

6.3 Tests

```
1 maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
3 Vous allez être redirigé vers l'esclave 2122
Tapez votre requete :
5 ls
backupServer
7 client
client.c
9 client.h
client.o
11 csapp.c
csapp.h
13 csapp.o
debug
15 Makefile
master
17 master.c
master.o
19 projetFTPBackup
rapport.pdf
21 rapport.tex
server
23 server.c
server.h
25 server.o
tell_wait.c
27 tell_wait.h
test
29 testt

31 Tapez votre requete :
pwd
33 /media/maxence/MyLinuxLive1/projetFTP
Tapez votre requete :
35 cd ..
Tapez votre requete :
37 ls
backup
39 cours
favoris.html
41 informatique
latex
43 Papier
projetFTP
45 projetFTPBackup
Revisions_S5
47 Revisions_S6-master
SAGA MP3
49 System Volume Information
t
51

53 Tapez votre requete :
pwd
/media/maxence/MyLinuxLive1
55

57 Tapez votre requete :
bye
fin de la connexion
59
```

Chapitre 7

Authentification

7.1 Objectif de la réalisation

Dans cette partie, nous avons mis en place un système d'authentification afin de protéger les fichiers gérés par le serveur. Pour cela, nous avons mis en place un mot de passe et un login, faisant échouer la connexion au serveur si erronée. Pour ce faire, nous disposons d'un fichier `.login` dans lequel se trouve à chaque ligne : `login : password`.

7.2 Code source commenté

7.2.1 master.c

```
1 int auth(char * login){
2     int b=0;
3     char buf[MAXLINE];
4
5     int fd = open(".login", O_RDONLY); //On ouvre le fichier comportant tous les login::password
6     rio_t rio;
7     Rio_readinitb(&rio, fd);
8
9     size_t n =0;
10
11     while((n = Rio_readlineb(&rio, buf, MAXLINE))>0 && !b){
12         //On lit chacune des lignes
13         //Si on lit le login envoye par le clientfd
14         //b prend la valeur 1 et on arrete la lecture
15         buf[n-1]='\0';
16         if(!strcmp(buf, login)){
17             b=1;
18             break;
19         }
20     }
21     return b;
22 }
23
```

7.3 Tests

```
2 maxence@Sybil:~$ ./client 127.0.0.1
3 client connected to server 127.0.0.1
4 login : erreur
5 Password : erreur
6 L'authentification a echouee
7 maxence@Sybil:~$ ./client 127.0.0.1
8 client connected to server 127.0.0.1
9 login : toto
10 Password : toto
11 Vous allez être redirigé vers l'esclave 2122
```

Chapitre 8

Les commandes rm, rm -r, mkdir et put

Nous n'avons pas réussi à implanter cette partie. Nous avons rencontré deux problèmes : tout d'abord nous avons un problème de bufferisation, lorsque nous utilisons une de ses commandes, si nous souhaitons ensuite utiliser la commande ls ou pwd, le serveur garde en mémoire le nom du fichier ou repertoire créé ou supprimer, par exemple après avoir taper : mkdir repTest, lorsque le client tape ls, le serveur cherche à exécuter ls repTest. Nous avons aussi un problème de communication maître/esclave. Néanmoins, si nous tapons la commande rm, rm -r ou mkdir, le serveur crée/supprime le fichier/repertoire concerné. Et la méthode put, téléverse le fichier.

8.1 code source commenté

8.1.1 server.c

Nous avons rajouté les options dans la fonction readRequest :

```
2  else if (!strcmp("put", req->cmd) || !strcmp("rm", req->cmd) || !strcmp("mkdir", req->cmd) ) {
//Si c'est une des requetes causant une modification des servers
3  if (!strcmp("rm", req->cmd)) {
4      char* args[3];
5      if (!strcmp(req->filename, "-r")) {
6          args[0] = "rm";
7          args[1] = "-r";
8          args[2] = req->content;
9      } else {
10         args[0] = "rm";
11         args[1] = req->filename;
12     }
13     int c;
14     if ((c=Fork())==0){
15         execvp(args[0], args);
16         exit(0);
17     }
18     waitpid(c, NULL, 0);
19 } if (!strcmp("mkdir", req->cmd)) {
20     char* args[2];
21     args[0] = "mkdir";
22     args[1] = req->filename;
23     int c;
24     if ((c=Fork())==0){
25         execvp(args[0], args);
26         exit(0);
27     }
28     waitpid(c, NULL, 0);
29 } else {
30     put(req);
31 }
32 }
```

Et nous avons rajouter une fonction put :

```
2  void put(struct Request* r) {
3      int l; ssize_t n=0;
4      rio_readn(req->connfd, &l, sizeof(int));
```

```

4      char * buf;
5      int fd = open(strcat(req->filename, "1"), O_RDWR | O_CREAT, 0666); //On creer un fichier pour
le tÃ©lÃ©chargement
6      buf = (char*) malloc(sizeof(char)*MAXSEND);
7      int send=MAXSEND;
8      if(l<MAXSEND)
9          send=l;
10     sleep(5);
11     while((n=r_readn(req->connfd, buf, send)) > 0 && l>0) { //Tant qu'on recoit des donnÃ©es, on
les ecrit dans le fichier
12         if (n>send)
13             n=send;
14         r_writen(fd, buf, n);
15         l-=n;
16         if(l<MAXSEND)
17             send=l;
18     }
19     close(fd);
20 }

```

8.1.2 client.c

Nous rajoutons les options suivantes pour client.c :

```

2      else if (!strcmp("put", req->cmd)){
3          //Si c'est une des requetes causant une modification des servers
4          stat(req->filename, &req->sbuf);
5          int l = req->sbuf.st_size;
6          r_writen(req->clientfd, &l, sizeof(int));
7          int fd = open(req->filename, O_RDONLY);
8          /*
9          Tant qu'on a ecrit, on continue d'ecrire
10         */
11         do {
12             n = r_readn(fd, buf, MAXSEND);
13             r_writen(req->clientfd, buf, n);
14         } while (n>0);
15         close(fd);
16     } else if (!strcmp("cd", req->cmd)) {}
17     else if (!strcmp("rm", req->cmd) || !strcmp("mkdir", req->cmd)) {}
18 }

```