Systèmes et Réseaux : Implémentation d'un pseudo serveur FTP

Grand Maxence, Muller Lucie (Groupe 1)

09/04/2017

Table des matières

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 récuperation 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{
     char * cmd; //La methode
3    char * filename; //Le fichier a prendre en commpte
     char * content; //informtion supplementaire
5    int connfd; //descripteur de socket
     int fini; //1 si bye envoye, 0 sinon
7    struct stat sbuf;
   };
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 récuperation de fichier à distance en donnant le nom du fichier á partir du client. Le serveur créé un nouveau fichier sur lequel, à l'aide d'un buffer, il copie le contenu du premier fichier. Ce nouveau fichier est sauvegardé a l'emplacement courant du client, sous le nom "nomdupremier fichier1"

1.2 Code source commenté

1.2.1 client.c

```
int main(int argc, char **argv)
{
  int clientfd, fd;
  char *buf, *host, *hidefile;
  struct timeval start, end;
  size_t len=0, n=0;

  if (argc != 2) {
    fprintf(stderr, "usage: %s <host>\n", argv[0]);
    exit(0);
  }
  host = argv[1];//On recupere le hostname du serveur (127.0.0.1 pour les tests)

  clientfd = Open_clientfd(host, port);//On demande la connexion au serveur

  req = malloc(sizeof(struct Request));//On alloue de la memoire pour la Request
  req->clientfd = clientfd;//On recupere le descripteur du socket

  Rio_readinitb(&rio, clientfd);//On initialise Rio

  printf("client connected to server %s\n", host);

  while (1) {
    printf("Tapez votre requete : \n");
    buf = (char*)malloc(sizeof(char)*MAXLINE);
    Fgets(buf, MAXLINE, stdin);
    req->filename = (char*)malloc(sizeof(char)*MAXLINE);
    req->cmd = (char*)malloc(sizeof(char)*MAXLINE);
    req->content = (char*)malloc(sizeof(char)*MAXLINE);
    sscanf(buf, "%s %s %s", req->cmd, req->filename, req->content);//On recupere la requette tapee
    par le client

    if(!strcmp(req->cmd, "get")){
      gettimeofday(&start, NULL);
      Rio_writen(req->clientfd, buf, strlen(buf));//On envoie la requete au serveur
      fd = open(strcat(req->filename, "1"), O_RDWR | O_CREAT, 0666);//On creer un fichier pour le
    tÃ©lÃ©chargement
      free (buf);
      buf = (char*)malloc(sizeof(char)*MAXSEND);
```

```
      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 echoÃ©e\n");
        remove(req->filename);
      }
    }else if(!strcmp(req->cmd, "bye")){//On ferme la connexion
      Rio_writen(req->clientfd, req->cmd, strlen(req->cmd));
      printf("fin de la connexion\n");
      exit(0);
    }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);
    }
  }
}
```

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);
```

```
                exit(0);
            }
            child[i] = p;
        }

        for(int i=0; i<NB_PROC; i++)
            waitpid(child[i], NULL, 0);
        //On attend tous les fils

        exit(0);
    }

    void handler(int sig){
      //On s'occupe des zombis
        waitpid(-1, NULL, WNOHANG|WUNTRACED);
        return;
    }
    void stop(int sig){
      //Si on recoit un SIGCTR, on tue tous les fils
        for(int i=0; i<NB_PROC; i++){
            kill(SIGKILL, child[i]);
        }
        exit(0);
    }

    void readRequest(struct Request *req){

        size_t n;
        req->cmd = (char*)malloc(sizeof(char)*MAXLINE);
        req->filename = (char*)malloc(sizeof(char)*MAXLINE);
        req->content = (char*)malloc(sizeof(char)*MAXLINE);
        char* request = (char *)malloc(sizeof(char)*MAXLINE);
        rio_t rio;
        Rio_readinitb(&rio, req->connfd);
        n=Rio_readlineb(&rio, request, MAXLINE);
        fflush(stdout);
        request[n-1] = '\0';

        sscanf(request, "%s %s %s", req->cmd, req->filename, req->content);
        //On recupere la requte , et remplis notre Request
        stat(req->filename, &req->sbuf);
        if(!strcmp("get", req->cmd)){
          get(req);
        }else if (!strcmp("bye", req->cmd) ) {
          printf("fin de la connexion");
          Close(req->connfd);
        }
        fflush(stdout);
    }

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


        //On envoie d'un coup
        int srcfd;
        char *srcp;
        //Rio_readinitb(&rio, fd);
        srcfd = Open(req->filename, O_RDONLY, 0);
        srcp = Mmap(0, req->sbuf.st_size, PROT_READ, MAP_PRIVATE, srcfd, 0);
        Close(srcfd);
        Rio_writen(req->connfd, srcp, req->sbuf.st_size);
        Munmap(srcp, req->sbuf.st_size);
    }
```

1.3 Tests

```
maxence@Sybil:~$ cat Makefile
CC=gcc
CFLAGS=-Wall
LIBS=-lpthread
all: server client
server: server.o csapp.o tell_wait.o
$(CC) $(CFLAGS) -o server server.o csapp.o tell_wait.o  $(LIBS)

client: client.o csapp.o
$(CC) $(CFLAGS) -o client  client.o csapp.o $(LIBS)
client.o: client.c csapp.c client.h csapp.h
$(CC) $(CFLAGS) -c client.c csapp.c $(LIBS)

server.o: server.c csapp.c server.h csapp.h
    $(CC) $(CFLAGS) -c server.c csapp.c $(LIBS)
tell_wait.o: tell_wait.c csapp.c tell_wait.h csapp.h
  $(CC) $(CFLAGS) -c tell_wait.c csapp.c $(LIBS)

csapp.o: csapp.c csapp.h
    $(CC) $(CFLAGS) -c csapp.c $(LIBS)
clean:
    rm server client *.o
maxence@Sybil:~$ ls
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
maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
Tapez votre requete :
get Makefile
617 bytes received in 0.002723 sec (226590.103143 bytes / sec)
Tapez votre requete :
bye
fin de la connexion
maxence@Sybil:~$ cat Makefile1
CC=gcc
CFLAGS=-Wall
LIBS=-lpthread

all: server client


server: server.o csapp.o tell_wait.o
  $(CC) $(CFLAGS) -o server server.o csapp.o tell_wait.o  $(LIBS)
client: client.o csapp.o
  $(CC) $(CFLAGS) -o client  client.o csapp.o $(LIBS)

client.o: client.c csapp.c client.h csapp.h
  $(CC) $(CFLAGS) -c client.c csapp.c $(LIBS)
server.o: server.c csapp.c server.h csapp.h
  $(CC) $(CFLAGS) -c server.c csapp.c $(LIBS)

tell_wait.o: tell_wait.c csapp.c tell_wait.h csapp.h
  $(CC) $(CFLAGS) -c tell_wait.c csapp.c $(LIBS)
csapp.o: csapp.c csapp.h
  $(CC) $(CFLAGS) -c csapp.c $(LIBS)

clean:
  rm server client *.o
```

Chapitre 2

Découpage du fichier

2.1 Objectif de la réalisation

Dans cette partie, nous avons du ameliorer 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 transferé. Pour cela on utilise une boucle qui envoie a 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

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

    int fd = open(req->filename, O_RDONLY);
    Rio_readinitb(&rio, fd);
    /*
  Tant qu'on a ecris, on continue d'ecrire
    */
    while ((n = Rio_readlineb(&rio, buf, MAXSEND)) > 0) {
        err = rio_writen(req->connfd, buf, n);
        if(err == -1){//On stoppe si le client s'arrete
            fprintf(stderr, "Arret innatendu du client\n");
            break;
        }
    }

  }

```

2.3 Tests

```
maxence@Sybil:~$ cat test
backupServer
client
client.c
client.h
client.o
csapp.c
csapp.h
csapp.o
Makefile
Makefile1
master.c
```

```
server
server.c
server.h
server.o
tell_wait.c
tell_wait.h
tell_wait.o
test
test_pipe.c
maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
Tapez votre requete :
get test
186 bytes received in 0.009943 sec (18706.611932 bytes / sec)
Tapez votre requete :
bye
fin de la connexion
maxence@Sybil:~$ cat test1
backupServer
client
client.c
client.h
client.o
csapp.c
csapp.h
csapp.o
Makefile
Makefile1
master.c
server
server.c
server.h
server.o
tell_wait.c
tell_wait.h
tell_wait.o
test
test_pipe.c
```

Chapitre 3

Gestion simple des pannes coté client

3.1 Objectif de la réalisation

Dans cette partie, nous devons gèrer les pannes du client, c'est-a-dire le comportement du serveur au cas où le client s'arrêterais 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, a 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 était 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

```
if(!strcmp(req->cmd, "get")){
  gettimeofday(&start, NULL);
  Rio_writen(req->clientfd, buf, strlen(buf));//On envoie la requete au serveur
  hidefile = (char*)malloc(sizeof(char)*MAX_NAME_LEN);
  hidefile[0]='T';
  strcat(hidefile, req->filename);//On creer le nom du fichier temporaire
  if(stat(hidefile,&req->sbuf) != 0){//Si le fichier temporaire n'existe pas
    open(hidefile, O_RDWR | O_CREAT, 0666);//On le cree
    fd = open(strcat(req->filename, "1"), O_RDWR | O_CREAT, 0666);//On cree un fichier pour le
    tÃ©lÃ©chargement
    free (buf);
    buf = (char*)malloc(sizeof(char)*MAXSEND);
    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 echoue\n");
      remove(req->filename);
    }
    remove(hidefile);
  }else{//Si le fichier temporaire existe
    fd = open(strcat(req->filename, "1"), O_WRONLY);
    stat(req->filename,&req->sbuf);
    int dejaLu = req->sbuf.st_size;//On recupere la taille des donnees deja telechargees
    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
maxence@Sybil:~$ head -3 test > test1
maxence@Sybil:~$ touch Ttest
maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
Tapez votre requete :
get test
186 bytes received in 0.000816 sec (227977.949737 bytes / sec)
Tapez votre requete :
bye
fin de la connexion
maxence@Sybil:~$ cat test1
backupServer
client
client.c
client.h
client.o
csapp.c
csapp.h
csapp.o
Makefile
Makefile1
master.c
server
server.c
server.h
server.o
tell_wait.c
tell_wait.h
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 a partager les differentes requêtes des clients entre plusieurs serveurs dit "esclaves". Cette répartition est gèrée par un serveur "maitre" qui reçoit toutes les requêtes avant de les envoyer aux esclaves. Lorsque le serveur maître reçoit une connexion vennant d'un client, il lui ernvoie 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;
2
3  int slave[NB_PROC];
4
5  int next(){//Renvoie le prochain esclave
6    int p = slave[sCourant];
7    sCourant = ++sCourant % NB_PROC;
8    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
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'esclve traitant
41
42       Close(connfd);//Ferme la connexion avec le client
```

```
    }
  }
  exit(0);
}

```

4.2.2 server.c

```
clientfd = Open_clientfd(host, port);//On demande la connexion au serveur maitre
printf("client connected to server %s\n", host);

Rio_readn(clientfd, &port2, sizeof(int)); //On recupert le port du serveur esclave

printf("Vous aller Ãªtre redirigÃ© vers l'esclave %d\n", port2);

clientfd = Open_clientfd(host, port2);//On demande la connexion au serveur esclave

req = malloc(sizeof(struct Request));//On alloue de la memoire pour la Request
req->clientfd = clientfd;//On recupere le descripteur du socket

```

4.3 Tests

```
maxence@Sybil:~$ ls > test
maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
Vous aller Ãªtre redirigÃ© vers l'esclave 2122
Tapez votre requete :
get test
238 bytes received in 0.004584 sec (51921.582857 bytes / sec)
Tapez votre requete :
bye
fin de la connexion
maxence@Sybil:~$ cat test
backupServer
client
client.c
client.h
client.o
csapp.c
csapp.h
csapp.o
debug
Makefile
master
master.c
master.o
projetFTPBackup
rapport.pdf
rapport.tex
rapport.toc
server
server.c
server.h
server.o
tell_wait.c
tell_wait.h
tell_wait.o
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 differentes requêtes d'un client sans fermer la connexion entre chacun d'elle. En d'autres termes il peut ainsi les gèrer les unes aprés les autres sans avoir a 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 :

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

```

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

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

5.3 Tests

```
maxence@Sybil:~$ ls > test
maxence@Sybil:~$ ls -l > testt
maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
Vous allez Ãªtre redirigÃ© vers l'esclave 2122
Tapez votre requete :
get test
262 bytes received in 5.005285 sec (52.344671 bytes / sec)
Tapez votre requete :
get testt
1969 bytes received in 5.008208 sec (393.154595 bytes / sec)
Tapez votre requete :
bye
fin de la connexion
maxence@Sybil:~$ cat test1
backupServer
```

```
client
client.c
client.c1
client.h
client.o
csapp.c
csapp.h
csapp.o
debug
Makefile
master
master.c
master.o
projetFTPBackup
rapport.aux
rapport.log
rapport.pdf
rapport.tex
rapport.toc
server
server.c
server.h
server.o
t
tell_wait.c
tell_wait.h
test
maxence@Sybil:~$ cat testt1
total 3456
-rwx------ 1 maxence maxence   3294 avril  1 11:41 backupServer
-rwx------ 1 maxence maxence  36488 avril  9 18:29 client
-rwx------ 1 maxence maxence   7025 avril  9 18:21 client.c
-rwx------ 1 maxence maxence   7018 avril  9 16:23 client.c1
-rwx------ 1 maxence maxence    224 avril  4 16:05 client.h
-rwx------ 1 maxence maxence   9952 avril  9 18:21 client.o
-rwx------ 1 maxence maxence  20259 mars  29 18:33 csapp.c
-rwx------ 1 maxence maxence   6105 mars  29 18:33 csapp.h
-rwx------ 1 maxence maxence  23760 avril  9 18:29 csapp.o
-rwx------ 1 maxence maxence      0 avril  9 16:45 debug
-rwx------ 1 maxence maxence    960 avril  9 16:24 Makefile
-rwx------ 1 maxence maxence  36704 avril  9 18:29 master
-rwx------ 1 maxence maxence   6867 avril  9 18:27 master.c
-rwx------ 1 maxence maxence  10136 avril  9 18:29 master.o
drwx------ 1 maxence maxence 131072 avril  9 18:14 projetFTPBackup
-rwx------ 1 maxence maxence   4046 avril  9 18:39 rapport.aux
-rwx------ 1 maxence maxence  44067 avril  9 18:39 rapport.log
-rwx------ 1 maxence maxence 259857 avril  9 18:39 rapport.pdf
-rwx------ 1 maxence maxence  22400 avril  9 18:39 rapport.tex
-rwx------ 1 maxence maxence   2538 avril  9 18:39 rapport.toc
-rwx------ 1 maxence maxence  36776 avril  9 18:29 server
-rwx------ 1 maxence maxence   6491 avril  9 18:28 server.c
-rwx------ 1 maxence maxence    452 avril  9 16:04 server.h
-rwx------ 1 maxence maxence  10728 avril  9 18:29 server.o
-rwx------ 1 maxence maxence   1794 fÃ©vr. 28 09:24 tell_wait.c
-rwx------ 1 maxence maxence    131 fÃ©vr. 28 08:55 tell_wait.h
-rwx------ 1 maxence maxence    262 avril  9 18:42 test
-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 créer un procecus 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

```
if (!strcmp("ls", req->cmd)){
    char* args[1];
args[0] = req->cmd;
int c;
int fd = open(".log", O_RDWR | O_CREAT, 0666);
if((c=Fork())==0){
    close(1);
    dup(fd);
    execvp(args[0], args);
    exit(0);
}
close(fd);
waitpid(c, NULL, 0);
memcpy(req->filename, ".log", 4);
stat(req->filename, &req->sbuf);
int l = req->sbuf.st_size;
rio_writen(req->connfd, &l, sizeof(int));
    get(req);
    remove(".log");
}else if (!strcmp("pwd", req->cmd)){
  char* args[1];
  args[0] = "pwd";
  int c;
  int fd = open(".log", O_RDWR | O_CREAT, 0666);
  if((c=Fork())==0){
      close(1);
    dup(fd);
    execvp(args[0], args);
    exit(0);
  }
  close(fd);
  waitpid(c, NULL, 0);
  memcpy(req->filename, ".log", 4);
  stat(req->filename, &req->sbuf);
  int l = req->sbuf.st_size;
  rio_writen(req->connfd, &l, sizeof(int));
  get(req);
  remove(".log");
}else if (!strcmp("cd", req->cmd)){
```

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

6.3 Tests

```
maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
Vous allez Ãªtre redirigÃ© vers l'esclave 2122
Tapez votre requete :
ls
backupServer
client
client.c
client.h
client.o
csapp.c
csapp.h
csapp.o
debug
Makefile
master
master.c
master.o
projetFTPBackup
rapport.pdf
rapport.tex
server
server.c
server.h
server.o
tell_wait.c
tell_wait.h
test
testt

Tapez votre requete :
pwd
/media/maxence/MyLinuxLive1/projetFTP
Tapez votre requete :
cd ..
Tapez votre requete :
ls
backup
cours
favoris.html
informatique
latex
Papier
projetFTP
projetFTPBackup
Revisions_S5
Revisions_S6-master
SAGA MP3
System Volume Information
t

Tapez votre requete :
pwd
/media/maxence/MyLinuxLive1

Tapez votre requete :
bye
fin de la connexion

```

Chapitre 7

Autentifcation

7.1 Objectif de la réalisation

Dans cette partie, nous avons mis en place un système d'authentification afin de protéger les fichiers gèrés 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

```
int auth(char * login){
    int b=0;
    char buf[MAXLINE];

    int fd = open(".login", O_RDONLY);//On ouvre le fichier comportant tous les login::passxord
    rio_t rio;
    Rio_readinitb(&rio, fd);

    size_t n =0;

    while((n = Rio_readlineb(&rio, buf, MAXLINE))>0 && !b){
        //On lit chacune des lignes
    //Si on lit le login envoye par le clientfd
    //b prend la valeur 1 et on arrete la lecture
    buf[n-1]='\0';
      if(!strcmp(buf, login)){
          b=1;
          break;
      }
    }
return b;
}

```

7.3 Tests

```
maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
login : erreur
Password : erreur
L'authentification a echouee
maxence@Sybil:~$ ./client 127.0.0.1
client connected to server 127.0.0.1
login : toto
Password : toto
Vous allez Ãªtre redirigÃ© vers l'esclave 2122

```

Chapitre 8

Les commandes rm, rm -r, mkdir et put

Nous n'avons pas réussi a implanter cette partie. Nous avons rencontré deux problèmes : tout d'abord nous avions 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 crér/supprime le fihcier/répertoire 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 :

```
  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
    if(!strcmp("rm", req->cmd)){
      char* args[3];
      if(!strcmp(req->filename, "-r")){
          args[0] = "rm";
          args[1] = "-r";
          args[2] = req->content;
      }else{
          args[0] = "rm";
          args[1] = req->filename;
      }
      int c;
      if((c=Fork())==0){
          execvp(args[0], args);
          exit(0);
      }
      waitpid(c, NULL, 0);
    }if(!strcmp("mkdir", req->cmd)){
      char* args[2];
      args[0] = "mkdir";
      args[1] = req->filename;
      int c;
      if((c=Fork())==0){
          execvp(args[0], args);
          exit(0);
      }
      waitpid(c, NULL, 0);

    }else{
      put(req);
    }
```

Et nous avons rejouter une fonction put :

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

```
    char * buf;
    int fd = open(strcat(req->filename, "1"), O_RDWR | O_CREAT, 0666);//On creer un fichier pour le tÃ©lÃ©chargement
    buf = (char*)malloc(sizeof(char)*MAXSEND);
    int send=MAXSEND;
    if(l<MAXSEND)
      send=l;
    sleep(5);
    while((n=rio_readn(req->connfd,buf, send)) > 0 && l>0) {//Tant qu'on recoit des donnÃ©es, on les ecrit dans le fichier
        if (n>send)
          n=send;
        rio_writen(fd, buf, n);
        l-=n;
        if(l<MAXSEND)
          send=l;
    }
    close(fd);
}
```

8.1.2 client.c

Nous rajoutons les options suivantes pour client.c :

```
else if (!strcmp("put", req->cmd)){
  //Si c'est une des requetes causant une modification des servers
  stat(req->filename, &req->sbuf);
  int l = req->sbuf.st_size;
  rio_writen(req->clientfd, &l, sizeof(int));
  int fd = open(req->filename, O_RDONLY);
  /*
  Tant qu'on a ecris, on continue d'ecrire
  */
  do {
      n = rio_readn(fd, buf, MAXSEND);
      rio_writen(req->clientfd, buf, n);
  }while (n>0);

  close(fd);
}else if(!strcmp("cd", req->cmd)){}
else if(!strcmp("rm", req->cmd) || !strcmp("mkdir", req->cmd)){}

```