Systèmes 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

```
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 tapÃ©e
    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 donnÃ©es, 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));
47    printf("%lu bytes received in %f sec (%f bytes / sec) \n",len, temps, ((double)(len/temps)));
    //On affiche le statistiques
      if(len == 0){
49      printf("Le tÃ©lÃ©chargement a echoÃ©e\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");
55    exit(0);
    }else{//Si la commande est inconnue on ferme la connexion
57    printf("Commande %s inconnue\n", req->cmd);
      printf("fin de la connexion\n");
59    exit(0);
    }
61  }
}
63
```

1.2.2 server.c

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

12      clientlen = (socklen_t)sizeof(clientaddr);

14      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 accept la connexion entrnte
20

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

24                      /* determine the textual representation of the client's IP address */
                        Inet_ntop(AF_INET, &clientaddr.sin_addr, client_ip_string,INET_ADDRSTRLEN)
    ;
26

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

32                      readRequest(req);//On lit la requete du client
                        Close(connfd);
34                  }
                }
36              freeRequest(req);
                exit(0);
38          }
            child[i] = p;
40      }

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

```
46          exit(0);
        }
48
        void handler(int sig){
50        //On s'occupe des zombis
            waitpid(-1, NULL, WNOHANG|WUNTRACED);
52          return;
        }
54      void stop(int sig){
          //Si on recoit un SIGCTR, on tue tous les fils
56          for(int i=0; i<NB_PROC; i++){
                kill(SIGKILL, child[i]);
58          }
            exit(0);
60      }

62      void readRequest(struct Request *req){

64          size_t n;
            req->cmd = (char*)malloc(sizeof(char)*MAXLINE);
66          req->filename = (char*)malloc(sizeof(char)*MAXLINE);
            req->content = (char*)malloc(sizeof(char)*MAXLINE);
68          char* request = (char *)malloc(sizeof(char)*MAXLINE);
            rio_t rio;
70          Rio_readinitb(&rio, req->connfd);
            n=Rio_readlineb(&rio, request, MAXLINE);
72          fflush(stdout);
            request[n-1] = '\0';
74
            sscanf(request, "%s %s %s", req->cmd, req->filename, req->content);
76          //On recupere la requte , et remplis notre Request
            stat(req->filename, &req->sbuf);
78          if(!strcmp("get", req->cmd)){
              get(req);
80          }else if (!strcmp("bye", req->cmd) ) {
              printf("fin de la connexion");
82            Close(req->connfd);
            }
84          fflush(stdout);
        }
86
        void get(struct Request *req){
88        char buf[MAXSEND];
          size_t n,err;
90        rio_t rio;


92
            //On envoie d'un coup
94          int srcfd;
            char *srcp;
96          //Rio_readinitb(&rio, fd);
            srcfd = Open(req->filename, O_RDONLY, 0);
98          srcp = Mmap(0, req->sbuf.st_size, PROT_READ, MAP_PRIVATE, srcfd, 0);
            Close(srcfd);
100         Rio_writen(req->connfd, srcp, req->sbuf.st_size);
            Munmap(srcp, req->sbuf.st_size);
102     }
```

1.3 Tests

```
maxence@Sybil:~$ cat Makefile
2 CC=gcc
CFLAGS=-Wall
4 LIBS=-lpthread

6 all: server client


8
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

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 stopper 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 coyé 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

```
/*
Pour savoir si on a dÃ©jÃ  tentÃ© de tÃ©lÃ©charger un fichier, on crÃ©er un fichier temporaire au dÃ©but du tÃ©lÃ©chargement
Lorsque le tÃ©lÃ©chargement est terminÃ© on le supprime
Si il existe un fichier temporaire associÃ© au fichier que lon veut tÃ©lÃ©charger
cela signifie qu'on a dÃ©jÃ  essayer de le tÃ©lÃ©charger, on tÃ©lÃ©charge uniquement les donnÃ©es manquante
*/
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 crÃ©er 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 crÃ©er
    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 donnÃ©es, 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 le statistiques
      if(len == 0){
      printf("Le tÃ©lÃ©chargement a echoÃ©e\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 donnÃ©es dÃ©jÃ  tÃ©lÃ©chargÃ©
    free (buf);
    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 ecrit si ce sont des donnÃ©es non tÃ©lÃ©chargÃ©es
          rio_writen(fd, buf, n);
44        free(buf);
          buf = (char*) malloc(sizeof(char)*MAXSEND);
46        len+=n;
        }else{//Sinon on dÃ©place 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 connection
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;
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
43     }
44   }
45   exit(0);
46 }
47
```

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

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

Autentifcation

8.1 Objectif de la réalisation

8.2 Code source commenté

8.3 Tests