Rapport de TP sur le mini-serveur HTTP

Maxime **Chambreuil** - Sébastien **Le Digabel**

16 janvier 2003

Table des matières

Chapitre 1

Détails sur nos travaux

1.1 But du TP

Le but de ce TP est de réaliser un mini-serveur HTTP.

Le client HTTP envoie une requête au serveur. Lorsqu'il recoit cette requête, le serveur crée alors une socket qui permettera le dialogue avec le client. Enfin, après analyse de la requête, le serveur revoie la page désirée.

1.2 La fonction openSocket()

```
int openSocket(char* srvr_addr, char* srvr_port)
```

Cette fonction crée une socket à l'adresse passée en paramètre :
– srvr_addr : Adresse du serveur
– srvr_port : Port du serveur
Elle retourne l'identifiant de la scoket créée. Pour cette fonction nous avons utilisé le support de cours sur les sockets pour bien assimiler le principe. Le reste n'a été que de la manipulation de code pour l'adapter aux circonstances.

1.3 La fonction getFileName()

```
char* getFileName(FILE* input_stream)
```

Cette méthode permet de retourner le nom du fichier désiré par le client lors de sa requête. La requête arrive sous cette forme :

"GET /page.html"

Par conséquent, on devrai découper cette chaîne de caractère pour ne récupérer que :

"page.html"

Nous utilisons donc les méthodes de manipulations de chaînes de caractères en C, telles que les fonctions *strdup()* ou encore *strsep()*
Nous avons d traiter deux cas supplémentaires :

- Le cas o le fichier n'existe pas : Erreur 404
- Le cas o le fichier n'est pas précisé : Par défaut index.html

Pour le cas o le fichier n'est pas précisé, il suffit juste de remplacer la chaîne de caractère correspondante au nom du fichier ("") par une chaîne contenant "index.html".

1.4 La fonction sendFile()

```
void sendFile(FILE* output_stream, char* file_name)
```

Cette fonction permet d'envoyer au client le fichier qu'il a souhaité obtenir lorsqu'il a saisi son url.
Son principe est assez simple, on récupère le nom du fichier désiré, on l'ouvre, et on envoie sur la socket de sortie ce fichier, *caractère par caractère...*

Pour le cas o le fichier n'existe pas, nous avons donc pris connaissance de sa non-existance via la fonction *access()* qui nous retourne un descripteur, et qui nous permet de savoir si celui ci est valide ou non. S'il est valide, pas de problème, s'il est invalide, c'est qu'il n'existe pas. Auquel cas le serveur renverra la chaîne suivante :
"404"
Chaîne que le client affiche pour avertir l'utilisateur que sa page n'est pas trouvée. On peut aisément remplacer "404" par une phrase un peu plus explicite...

1.5 Conclusion

Cela marche très très bien, on va pouvoir maintenant programmer le remplacant de Mozilla :)

Chapitre 2

Le code

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

static void bail(const char *on_what) {
  if ( errno != 0 ) {
    fputs(strerror(errno),stderr);
    fputs(": ",stderr);
  }
  fputs(on_what,stderr);
  fputc('\n',stderr);
  exit(1);
}



/*
  openSocket() retourne
    l'identifiant d'un socket associ\'e \`a l'adresse IP srvr_addr:srvr_port
```

```
    -1 si impossibilit? de cr\'eer le socket
*/

int openSocket(char* srvr_addr,char* srvr_port) {

  int socketid = 0;
  struct sockaddr_in socketAdresse;
  int taille;
  int test;

  // On met les parametres par defaut si y a besoin
  if(srvr_addr == "")
    srvr_addr = "127.0.0.1";
  if (srvr_port == "")
    srvr_port = "8080";

  // On cree le socket reseau, TCP
  socketid = socket(PF_INET,SOCK_STREAM,0);
  if ( socketid == -1)
    bail ("socket()");

  // On reserve l'espace memoire pour les parametres du socket
  memset(&socketAdresse,0,sizeof socketAdresse);

  // On renseigne les differents champs
  socketAdresse.sin_family = AF_INET;
  // L'adresse IP
  socketAdresse.sin_addr.s_addr = inet_addr(srvr_addr);
  // Le port TCP
  socketAdresse.sin_port = htons (atoi(srvr_port));
  // Taille de socketAdresse
  taille = sizeof socketAdresse;

  // On specifie les parametres au socket
  test = bind (socketid,(struct sockaddr *) &socketAdresse,taille);
  if ( test == -1 )
    bail("bind()");

  // On retourne l'identifiant du socket
  return socketid;
}
```

```
/*
  Sachant que input_stream contient une chaine du type   GET /[nom_du_fichier] H
  getFileName() retourne :
     nom_du_fichier si celui ci est precis\'e
     index.html sinon
*/

char* getFileName(FILE* input_stream) {

  int i;
  char* nomfichier;

  /* initialisation d'une variable temporaire \`a 0 pour
     \'eviter les erreurs de fin de cha\^ines de caract\`ere */
  char* temp = malloc(150*(sizeof(char)));

  for(i=0;i<150;i++)
    temp[i]=0;

  // On recupere la chaine de caractere
  fgets(temp,150,input_stream);

  // On enleve le debut : "GET /" soit 5 caracteres
  temp = strdup(temp+5);

  // On recupere le nom de la ressource : la chaine avant le premier espace
  nomfichier = strsep (&temp," ");

  // On teste si la chaine est vide
  if (strcmp(nomfichier,"")==0)
    strcpy(nomfichier,"index.html");

  // on affiche du ct\'e serveur le nom du fichier qui sera retourn\'e
  printf ("nomfichier = \%s\n",nomfichier);

  // On retourne le nom du fichier
  return nomfichier;
```

```
}



/*
  sendFile() copie les donn?es contenues dans le fichier r?f?renc? par file_name
  sinon ?crit un message d'erreur 404  dans output_stream
*/

void sendFile(FILE* output_stream, char* file_name) {
  FILE* descripteurFichier;
  char caractereLu;
  int test;

  // Variable ne servant qu'a la copie ligne a ligne
  // char* chaineLu;

  // Ouverture du fichier
  descripteurFichier = fopen(file_name,"rb");

  // Si le fichier n'existe pas, on retourne une erreur 404
  if (access(file_name, F_OK)<0)){
    fputc('4',output_stream);
    fputc('0',output_stream);
    fputc('4',output_stream);
  }

  // Si le fichier existe
  else {
    do {
      // on lit dans le fichier
      caractereLu = fgetc(descripteurFichier);

      // on copie caractere a caractere dans output_stream
      fputc(caractereLu,output_stream);

      /* Code pour la copie ligne a ligne(ne marche pas)
         fgets(chaineLu,200,descripteurFichier);
         gputs(output_stream,chaineLu);
        while(chaineLu != NULL);
```

```
      */
    }
    // jusqu'a ce qu'on rencontre le caractere de fin de fichier
    while(caractereLu != EOF);
  }
  // On ferme le fichier
  fclose(descripteurFichier);
}

void do_process(int client_socket) {

  FILE* output_stream;
  FILE* input_stream;
  char* file_name;

  input_stream=fdopen(client_socket,"r");
  if (input_stream!=NULL)  {
    output_stream=fdopen(dup(client_socket),"w");
    if (output_stream!=NULL)   {
      file_name=getFileName(input_stream) ;
      if (file_name!=NULL) {
        sendFile(output_stream,file_name);
        free(file_name);
      }
      fclose(input_stream);
      fclose(output_stream);
    } else {
      fclose(input_stream);
    }
  }
}

int main(int argc,char **argv) {
  int socket = -1;
  int client_socket = -1;
  pid_t PID;
  int len_inet;
  struct sockaddr_in adr_clnt;

  socket=openSocket("127.0.0.1","8080");
  if ( socket == -1 )
```

```
    bail("openSocket()");

  if ( listen(socket,10)==-1 )
    bail("listen()");

  for (;;) {
    len_inet = sizeof adr_clnt;
    client_socket = accept(socket,(struct sockaddr *) &adr_clnt, &len_inet);
    if ( client_socket == -1 )
      bail("accept()");
    if ( (PID = fork()) == -1 ) {
      close(client_socket);
      continue;
    } else if ( PID > 0 ) {
      close(client_socket);
      continue;
    }
    do_process(client_socket);
    exit(0);
  }
  close(socket);
  return 0;
}
```