

Manuel Technique

Projet réseau : un « petit » client ftp

**Maquaire Myriam
Jilibert Laurent
L3 GMI**

Sommaire

1) Aperçu du protocole FTP.....	3
2) Historique.....	4
3) Description de la structure de donnée.....	5
4) Spécification des fonctions écrites.....	6
4.1) Fichier my_tools.c.....	6
4.2) Fichier Client.c.....	10
4.3) Fichier my_socket.c.....	10
4.4) Fichier my_ctrl.c.....	10
5) Lancement.....	11
6) Le proxy.....	11
Bibliographie.....	12

1) Aperçu du protocole ftp

Le protocole ftp permet l'échange fiable de fichiers à travers un réseau TCP. Il utilise deux types de connexions entre le serveur et le client :

- Une connexion permanente pour l'envoi de commandes du client au serveur et l'envoi des réponses du serveur au client. Nous désignerons cette connexion par le terme de « connexion de commandes ». Cette connexion utilise le protocole telnet [PR83].
- Des connexions pour l'envoi des données que ce soit du client vers le serveur (par ex, pour déposer un fichier sur le serveur) ou du serveur vers le client (par ex, pour récupérer des fichiers). Nous désignerons ces connexions par les termes de « connexion de données » ou encore « canal de données ».

Ainsi, une session ftp classique peut se décomposer comme suit :

- Etablissement de la connexion de commande par le client.
- Envoi des commandes de « login/password » et lecture des réponses du serveur.
- Exécution des commandes d'échange de fichier.
- Fermeture de la connexion de commande.

Nous n'implémenterons pas toutes les fonctionnalités offertes par le protocole ftp. Les commandes ftp que nous aurons à gérer seront de deux types :

- 1^{er} type de commande :
 - le client envoie une commande via le canal de commande
 - le serveur lui renvoie sa réponse par ce même canal
- 2^{ème} type de commande :
 - le client précise le type de connexion de données à utiliser
 - le serveur lui renvoie sa réponse par ce même canal
 - le client envoie une commande au serveur via le canal de commande
 - le serveur lui confirme le commencement de l'exécution de la commande via la connexion de commande
 - une connexion de données est établie
 - les données sont envoyées (par le client ou le serveur selon le besoin)
 - le canal des données est fermé par l'émetteur des données
 - le serveur envoie au client une réponse de bonne exécution de la commande via la connexion de commande

Les lignes sont par défaut terminées par les caractères \r\n.

2) Historique

FTP a subi une grande évolution au fil des ans. L'appendice III est une compilation chronologique des RFC se rapportant à FTP. Elle inclue la première proposition de mécanisme de transfert de fichiers de 1971 qui avait été développée pour une application sur les hôtes du M.I.T. (RFC 114), plus des commentaires et discussions dans la RFC 141. La RFC 172 proposait un protocole de niveau utilisateur pour le transfert de fichiers entre ordinateurs (y compris des terminaux IMPs). Une révision de celui-ci (RFC 265), redonnait un état du FTP pour évolution ultérieure, tandis que la RFC 281 suggérait encore d'autres modifications. L'usage d'une transaction "Set Data Type" a été proposée dans la RFC 294 en Janvier 1982. La RFC 354 a rendu les RFC 264 et 265 obsolètes. Le File Transfer Protocol était désormais défini comme un protocole de transfert de fichiers entre des hôtes d'un ARPANET, et dont la fonction première était définie comme le transfert efficace et fiable entre des hôtes pour profiter de l'utilisation d'une capacité de stockage de données distante. La RFC 385 apporte un correctif à certaines erreurs, développe certains points, et ajoute certaines notions au protocole, tandis que la RFC 414 définit le rapport d'état sur le serveur de travail et les "clients" FTP. La RFC 430 de 1973, (parmi d'autres trop nombreuses pour être mentionnées toutes) donnait des commentaires supplémentaires quant à FTP. Finalement, une documentation "officielle" FTP a été publiée sous la référence RFC 454.

Depuis Juillet 1973, des changements considérables sont intervenus, mais la structure globale est restée la même. La RFC 542 a été publiée comme une nouvelle spécification "officielle" pour refléter certains changements. Cependant, de nombreuses implémentations basées sur l'ancienne spécification n'étaient pas remises à jour. En 1974, les RFC 607 et 614 apportent de nouveaux commentaires à propos de FTP. La RFC 624 propose des changements nouveaux et autres modifications mineures. En 1975, la RFC 686 intitulée, "Leaving Well Enough Alone" était une discussion sur les différences entre toutes les anciennes versions de FTP et la dernière en date. La RFC 691 est une révision mineure de la RFC 686, concernant les possibilités d'impression de fichiers.

Motivée par le passage du NCP (Network Communication Protocol) à TCP comme protocole sous-jacent, un phoenix est rené à partir de tous les efforts ci-dessus par la RFC 765 comme une nouvelle spécification de FTP basée sur le protocole réseau TCP.

3) Description de la structure de donnée

Voici la structure de donnée utilisée dans notre application ftp. Cette structure permet de prendre comme variable un "socket_ctrl" qui représente le socket associé au canal de commande et un "socket_data" qui représente le socket associé au canal de données.

Le nom de serveur est représenté par la variable "server[255]". Cette variable peut donc prendre au maximum 255 caractères. Elle peut contenir soit une adresse IP ou un nom de serveur : quelque chose qui identifie le serveur.

La variable "port" représente le port, soit 20 soit 21.

La variable "mode" représente le mode connexion c'est-à-dire le mode passif ou le mode actif. Ces 2 modes de connexions sont associés à des variables statiques "PASSIF 0" et "ACTIF 1".

La variable "type" représente le type des données qui sont envoyées via le canal de données, c'est-à-dire "IMAGE" ou "ASCII".

Pour une question de facilité d'utilisation et de compréhension de programmation, on définit la notion de booléen à travers l'énumération "BOOL" qui peut prendre la valeur false ou true.

Enfin, une taille arbitraire de buffer a été choisie afin d'éviter les déplacements de capacité.

Structure se situant dans my_struct.h :

```
#ifndef MY_STRUCT_H
#define MY_STRUCT_H

#define BUFSIZE 1024
#define PASSIF 0
#define ACTIF 1

typedef struct ftp_socket{
int socket_ctrl;
int socket_data;
char server[255];
int port;
int mode;
int type;
} ftp_socket;

/*Declaration du type boolean*/
typedef enum {FALSE,TRUE} BOOL;

#endif
```

4) Spécification des fonctions écrites

Notre ftp est composé de 4 fichiers .c possédant tous un fichier header. A ces fichiers s'ajoute le fichier my_struct.h vu précédemment ainsi qu'un makefile.

- my_tools.c : fonctions associées aux commandes ftp.
- Client.c : le main, initialisation de la structure et vérifications des paramètres.
- my_socket.c : création et gestion de la socket.
- my_ctrl.c : opérations sur la socket comme l'envoi et la réception de messages (voir donnée et fichier aussi), contrôle les opérations pour la création du PASV (recup de l'ip envoyé par le serveur et traitement) et du PORT (envoi de notre ip et de notre port, gestion de la socket, bind, listen et accept)

4.1) Fichier my_tools.c

La commande USER permet d'entrer le nom du compte sous lequel on veut ouvrir une session. Commande de 1^{er} type.
(L'analyse des autres fonctions sera plus succincte)

```
int user(int sock, char *nom) {
    char *tmp = (char*)malloc(20*sizeof(char));
    char *txt = (char*)malloc(40*sizeof(char));
    int val;

    sprintf(txt, "USER %s", nom);

    sendCtrl(sock, txt);

    val = recvCtrl(sock);
    if( val != 331 ) {
        return -2;
    }

    /* PASSWORD */
    printf("> Password : ");
    scanf("%s",tmp);

    sprintf(txt, "PASS %s", tmp);
    sendCtrl(sock, txt);

    val = recvCtrl(sock);
    if( val != 230 ) {
        return -3;
    }
    return val;
}
```

Tout d'abord l'utilisateur doit saisir son nom d'utilisateur (user) puis son mot de passe. Différents tests sont effectués sur les réponses du serveur après l'envoi par le client des différentes chaînes de caractères saisies au travers du mot de passe et du nom d'utilisateur.

Les codes de réponse sont constitués de 3 chiffres dont voici les significations :

- Le premier chiffre indique le statut de la réponse (succès ou échec)
- Le second chiffre indique ce à quoi la réponse fait référence
- Le troisième chiffre donne une signification plus spécifique (relative à chaque deuxième chiffre)

Le code 331 correspondant au fait des informations supplémentaires sont demandées au client (ici la saisie du mot de passe) et donc que l'action demandée est en suspend. Mais aussi ce code convient de "l'authentification et compte" ce qui correspond bien à la saisie requise dans la fonction USER.

Dans la mesure où les différents codes sur lesquels sont effectués les tests ne sont pas rencontrés alors l'utilisateur est renvoyé au point de départ.

Premier chiffre		
Chiffre	Signification	Description
1yz	Réponse préliminaire positive	L'action demandée est en cours de réalisation, une seconde réponse doit être obtenue avant d'envoyer une deuxième commande
2yz	Réponse positive de réalisation	L'action demandée a été réalisée, une nouvelle commande peut être envoyée
3yz	Réponse intermédiaire positive	L'action demandée est temporairement suspendue. Des informations supplémentaires sont attendues de la part du client
4yz	Réponse négative de réalisation	L'action demandée n'a pas eu lieu car la commande n'a temporairement pas été acceptée. Le client est prié de réessayer ultérieurement
5yz	Réponse négative permanente	L'action demandée n'a pas eu lieu car la commande n'a pas été acceptée. Le client est prié de formuler une requête différente

Second chiffre		
Chiffre	Signification	Description
x0z	Syntaxe	L'action possède une erreur de syntaxe, ou bien il s'agit d'une commande non comprise par le serveur
x1z	Information	Il s'agit d'une réponse renvoyant des informations (par exemple pour une réponse à une commande STAT)
x2z	Connexions	La réponse concerne le canal de données
x3z	Authentification et comptes	La réponse concerne le login (USER/PASS) ou la demande de changement de compte (CPT)
x4z	Non utilisé par le protocole FTP	
x5z	Système de fichiers	La réponse concerne le système de fichiers distant

Figure 1 : Code des réponses FTP

`Char * readLine() ;`

Permet de lire une ligne entière, c'est-à-dire sans avoir de problème (du au scanf).

`char **str_split(char *cmd, char *delim);`

Permet de découper une chaîne suivant un délimiteur.

`int makeCmd(ftp_socket socket, char *chaine);`

Permet d'exécuter une commande.

`int cd(int socket, char *dir);`

Permet de se déplacer du répertoire courant vers le répertoire spécifié en paramètre.

`int delete(int socket, char *file);`

Permet de supprimer un fichier du serveur.

`int getfile(ftp_socket socket, char *file);`

Retourne un entier permettant de savoir si la réception du fichier s'est bien effectué (226) ou non (-1).

`int help();`

Permet d'afficher une explication sur chaque commande de notre ftp.

`int ls(ftp_socket socket, char *dir);`

Etablit une liste des fichiers et répertoires du répertoire distant.

`int mkdirec(int socket, char *dir);`

Permet de créer un répertoire sur le serveur.

`int mode(ftp_socket sock);`

Permet de changer de mode de transfert.

En effet il y a deux modes de transfert de données :

- mode Actif : c'est le client FTP qui détermine le port de connexion à utiliser pour permettre le transfert des données
- mode Passif : le serveur FTP détermine lui même le port de connexion à utiliser pour permettre le transfert des données (data connexion) et le communique au client

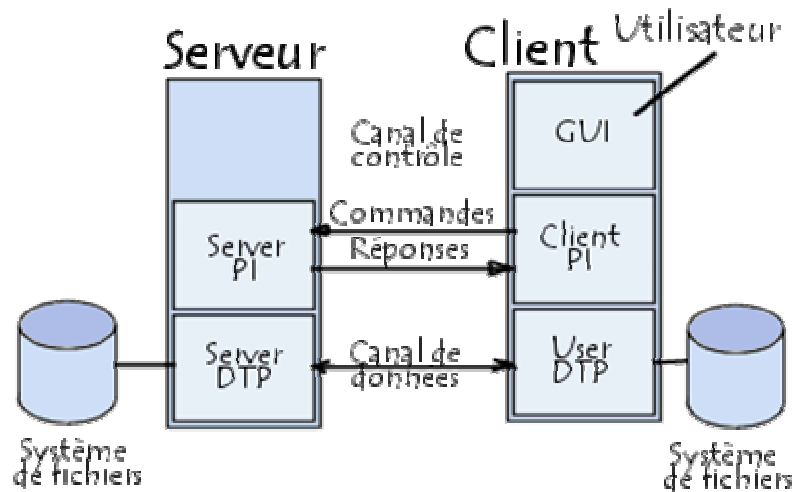


Figure 2 : modèle ftp

```
int put(ftp_socket socket, char *file);
```

Permet de stocker un fichier sur le serveur. Comme get, cela renvoie ou 226 ou -1.

```
int pwd(int socket);
```

Permet d'afficher le chemin du répertoire courant.

```
int rmdirec(int socket, char *dir);
```

Permet de supprimer un répertoire sur le serveur.

```
int syst(int socket);
```

Permet d'afficher le système d'exploitation du serveur.

```
int type(ftp_socket socket)
```

Permet de changer le type de représentation des données qui seront envoyés via le canal de données. Les types sont A (pour ASCII) et I (pour image).

```
void ftpLog();
```

Permet de créer un fichier de log.

```
void ftpTrace(const char *fmt, ...);
```

Permet d'écrire dans le fichier de log.

```
void ftpCloseLog();
```

Permet de fermer le fichier de log.

4.2) Fichier Client.c

`int runID();`

Permet de lancer l'authentification de l'utilisateur en lui demandant son login et son mot de passe. Si l'utilisateur est reconnu par les bases du serveur, ftpShell est lancé.

`void ftpShell();`

Permet d'interpréter des commandes.

4.3) Fichier my_socket.c

`int createSocket(ftp_socket sock, int p);`

Permet de créer une socket sur le port p.

4.5) Fichier my_ctrl.c

`int recvCtrl(int sock);`

Permet de recevoir sur la socket sock les messages envoyés par le serveur et les afficher sur la sortie standard.

`int recvJust(int sock);`

Permet de faire la même chose que la fonction précédente mais n'affiche pas sur la sortie standard.

`int sendCtrl(int sock, char *mess);`

Permet d'envoyer une commande qui sera interprété par l'application sur la socket sock.

`void recvData(int sock);`

Permet de recevoir les données sur la socket sock.

`void recvFile(int sock, char *file);`

Permet de recevoir le fichier sur la socket.

`void sendFile(int sock, char *file);`

Permet d'envoyer un fichier sur une socket.

```
int makePasv(ftp_socket sock);
```

Permet d'établir une connexion en mode passif. La socket data dans la structure prend la valeur renvoyé par PASV.

```
int makePort(ftp_socket sock);
```

Permet d'établir une connexion en mode actif. Envoi de la commande port avec le numéro de port et l'adresse du serveur. Création de la socket avec tout les traitements que cela inclus : bind, listen (mais pas accept).

```
int acceptMe(int desc);
```

Permet d'accepter une demande de connexion du serveur sur la socket DESC.

Par défaut, notre ftp est en mode passif et donc connecté au port 21.

5) Lancement

Le lancement de notre ftp se fait à partir de notre fonction main() qui se situe dans le fichier Client.c. L'utilisateur doit se loguer.

6) Le proxy

Le proxy on le passe quand le client fait un connect lors de l'établissement de la socket mais aussi lorsqu'on ne choisit que des ports compris 1100 et 1150 ce qui a pour conséquence que cela passe le proxy d'un serveur HTTP.

Nous avons donc réaliser un client ftp minimal qui permet de télécharger et déposer des fichiers sur le serveur ainsi que de parcourir l'arborescence du serveur.

Bibliographie

- <http://www.commentcamarche.net/>
- <http://www.iprelax.fr/ftp>
- <http://www.developpez.com/>
- Programmation Système en C sous Unix, Christophe Blaess
- cours de Mr Caron
- man

Merci à Mr Guesnet et Mr Caron pour leurs explications et leurs aides.