Travaux pratiques de programmation système et réseau (troisième année EUDIL, option informatique). Réalisation d'un client FTP.

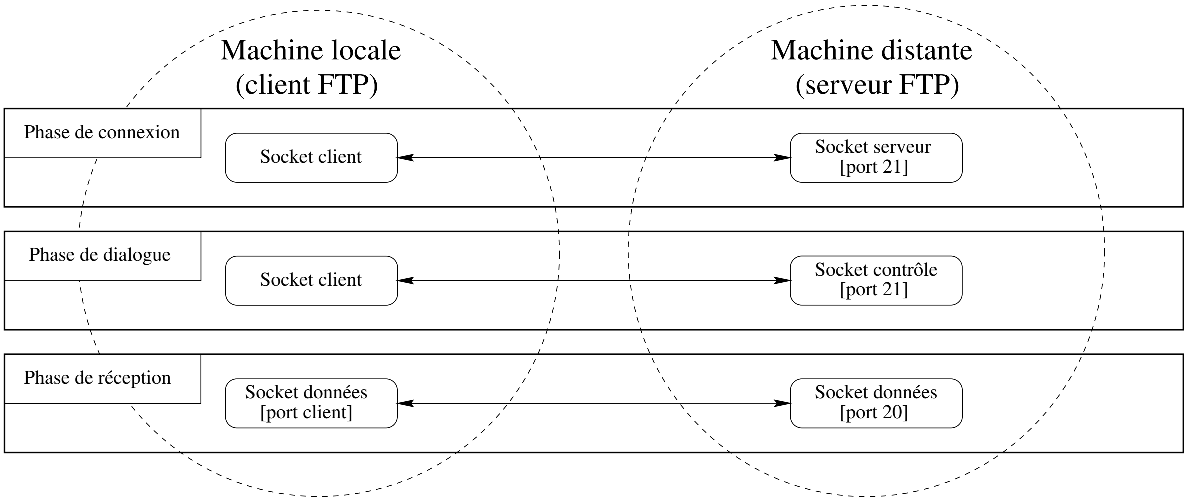
Xavier Redon

25 septembre 1995

1 Description du protocole FTP

Le but de ces travaux pratiques est la construction d'un client FTP.

Le protocole FTP (File Transfert Protocol) est utilisé pour échanger des fichiers avec une machine distante. Il s'agit d'un protocole implanté suivant le principe client-serveur, la figure ci-dessous en illustre les différentes phases :



Un processus serveur FTP tourne sur la machine distante et écoute, via une socket, un port TCP donné (le port 21 est alloué à FTP). Lorsqu'une machine cliente se connecte sur ce port, une socket dite de contrôle est établie entre les deux machines. Via cette socket le client dialogue avec le serveur et demande éventuellement un transfert de fichier. Les transferts de fichiers (et les listes de répertoires) ne se font pas par la socket de contrôle mais par une autre socket dite de données (par défaut, sur le serveur il s'agit du port 20, et sur le client du port déjà utilisé par la socket de contrôle). Pour la réception de fichier, la machine cliente doit être en attente de connexion sur la socket de données. Le serveur se connecte alors sur cette socket et le transfert du fichier (ou de la liste de répertoire) débute.

La conversation entre le client et le serveur a lieu en "clair", en voici un exemple (les messages de la machine serveur commence toujours par un nombre à trois chiffres) :

```
220 FTP server ready.
USER anonymous
```

```
331 Guest login ok, send ident as password.
PASS user@adresse
230 Guest login ok, access restrictions apply.
TYPE I
200 Type set to I.
CWD pub
250 CWD command successful.
PWD
257 "/pub" is current directory.
PORT a3,a2,a1,a0,p1,p0
200 PORT command successful.
LIST
150 Opening data connection for /bin/ls (a,p) (0 bytes).
226 Transfer complete.
QUIT
221 Goodbye.
```

Lors de cette session le client a effectué les actions suivantes :

1. identification en tant qu'invité,
2. spécification du mode binaire comme mode de transfert (mode `I`),
3. changement de répertoire courant,
4. demande du nom du répertoire courant,
5. spécification de l'adresse de la socket de données (l'adresse TCP `(a,p)` est donnée octet par octet en commençant par les octets de poids fort, `a` représente l'adresse IP et `p` le port)
6. demande de transfert de la liste du répertoire courant,
7. fermeture de la connexion.

Pour plus de détails sur le protocole FTP consultez le RFC 959.

2 Objectif du TP

Il est possible de dialoguer directement avec le serveur FTP via un `telnet` sur le port 21 :

```
telnet ftp.univ-lille1.fr 21
Trying 134.206.1.36 ...
Connected to graffiti.univ-lille1.fr.
Escape character is '^]'.
220 graffiti.univ-lille1.fr FTP server ready.
```

Cependant c'est un moyen assez "fruste" de rapatrier des fichiers c'est pourquoi il existe des clients FTP qui font interface entre l'utilisateur et le serveur FTP.

2.1 Exemple d'utilisation du client FTP

Le client FTP que vous allez réaliser est assez inhabituel dans la mesure où il masque l'aspect "orienté connexion" du protocole FTP. Un client classique autorise l'exploration d'un serveur puis, seulement après déconnexion, l'exploration d'un autre serveur. Votre serveur doit permettre l'exploration simultanée de plusieurs sites FTP, comme dans l'exemple ci-dessous :

```
$ ftpcd //ftp.univ-lille1.fr/pub
$ ftpls //ftp.univ-lille1.fr

gnu
faq
unix
X
pc
vms
mac
documentation
NOUVEAU
images
mpeg
sounds
www

$ ftpls //ftp.univ-poitiers.fr/pub/pc

internet
telnet

$ ftpcd //ftp.univ-poitiers.fr/pub/pc/telnet
$ ftpls //ftp.univ-poitiers.fr

install.txt
readme.txt
tel2308b.zip
tel2308s.zip

$ ftpcat //ftp.univ-lille1.fr/NOUVEAU | head

#
############################################################################
#                               Bonjour a vous !
#                           Amateurs de nouveautes
#                     Utilisateurs impatients de l'INTERNET
############################################################################

$ ftpcat //ftp.univ-poitiers.fr/readme.txt | head

NCSA Telnet 2.3.08
This is a maintenance release of NCSA Telnet for PCs running MS-DOS.

BUGS FIXED for NCSA Telnet

Help screen keys limited to avoid paramater menu.
```

2.2 La syntaxe des commandes du client

Comme le montre l'exemple précédent trois commandes sont suffisantes pour explorer un site FTP et rapatrier des fichiers :

- une commande `ftpcd` pour changer de répertoire courant sur un serveur FTP,
- une commande `ftpls` pour lister un répertoire sur un serveur FTP,
- une commande `ftpcat` pour afficher un fichier d'un serveur FTP.

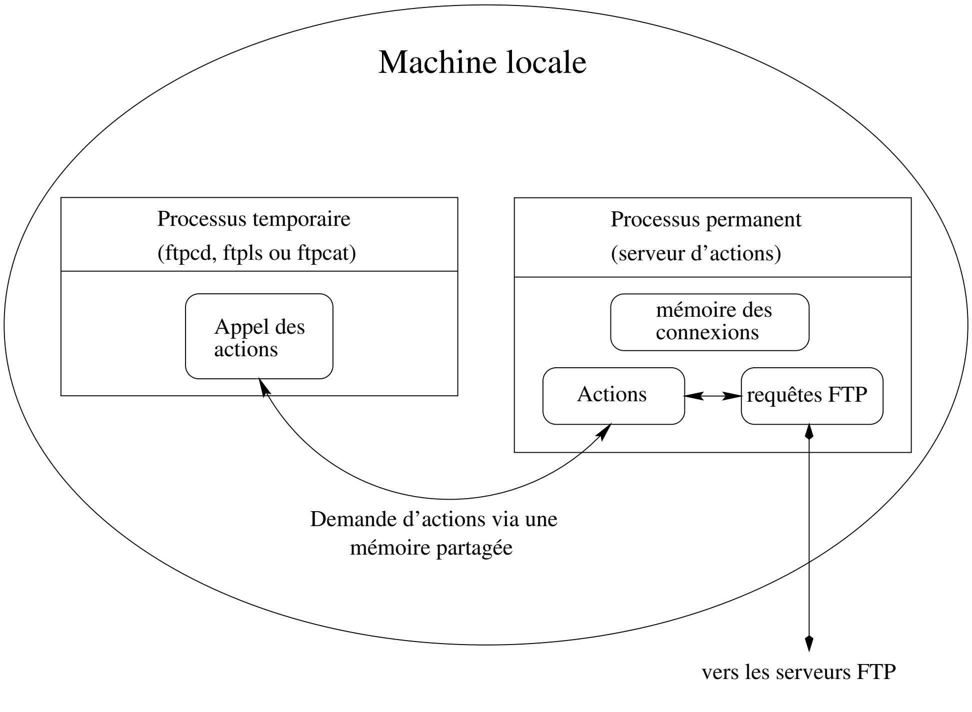
La convention utilisée pour désigner les fichiers et les répertoires consiste à coller le nom du serveur et le nom du fichier sous la forme `//serveur/fichier`.

2.3 L'architecture du client FTP

Pour que le client FTP soit efficace, il est demandé de minimiser les phases de connexion et de déconnexion. Ceci peut être implanté facilement en gardant en mémoire la liste des connexions déjà établies. Lorsqu'une des commandes `ftpcd`, `ftpls` ou `ftpcat` souhaite converser avec un serveur FTP elle doit appliquer l'algorithme ci-dessous :

- la connexion a-t-elle déjà été établie ?
  - si oui, la connexion est-elle toujours active (tester avec un `NOOP`) ?
    - si oui, converser directement avec le serveur.
    - si non, se connecter à nouveau au serveur.
  - si non, se connecter au serveur, stocker les références de cette connexion.

Il n'est pas possible de mémoriser les connexions au niveau des processus `ftpcd`, `ftpls` ou `ftpcat` car ils ont une durée de vie très courte. Une solution est d'introduire un troisième processus pour garder trace des connexions. L'architecture du client FTP est décrite dans le schéma ci-dessous :



Dans cette architecture c'est le processus serveur d'actions qui dialogue avec les serveurs FTP et qui garde les connexions (i.e. l'adresse IP du serveur et la socket de contrôle) en mémoire. Les demandes d'actions se font via une mémoire partagée protégée par un sémaphore d'exclusion mutuelle. Les différentes actions à implanter sont les suivantes :

- La connexion à un serveur FTP. Cette connexion consiste non seulement à établir la socket de contrôle mais aussi à s'identifier comme étant un invité (i.e. l'utilisateur `anonymous` avec le mot de passe `user@` où `user` est le nom de l'utilisateur local qui a lancé le processus serveur d'actions). De plus on passe en mode de transfert binaire pour ne pas avoir de problème avec les fichiers compressés et les exécutables. Bien entendu, cette connexion ne doit se faire réellement qu'une seule fois pour un serveur donné, ensuite on se contentera de chercher la socket de contrôle dans la table des connexions et de vérifier que le serveur répond toujours.
- Le changement de répertoire courant sur un serveur FTP donné. La connexion avec le serveur est sensée être établie.
- La réception de liste de répertoire. C'est une opération assez complexe qui doit se faire en plusieurs étapes :
  1. création d'une nouvelle socket en attente de connexion ;
  2. indication de l'adresse de cette socket au serveur FTP à l'aide de la commande `PORT` ;
  3. initiation du transfert à l'aide de la commande FTP `NLST` ;
  4. enfin, sauvegarde des données arrivant sur la nouvelle socket.

  On suppose là encore que la connexion avec le serveur est déjà établie.
- La dernière action à prévoir est la réception de fichier. C'est une action très similaire à la réception de liste, il serait même bon de prévoir une fonction générique. Il faut simplement utiliser la commande FTP `RETR` au lieu de `LIST`.

3 Travail à réaliser

3.1 Exercices de manipulation de FTP

Avant de passer à la programmation proprement dite, quelques exercices vous permettront de mieux comprendre le fonctionnement du protocole FTP.

3.1.1 Utilisation de `netstat`

Utilisez `netstat` pour vérifier qu'un serveur FTP est disponible sur votre machine. Quel est l'état de la socket de ce serveur ? Connectez-vous sur votre machine avec FTP (utilisez votre compte utilisateur). Que se passe-t-il au niveau des sockets de la machine ? Effectuez une deuxième connexion. Conclusion ? Plus difficile : transférez un fichier et pendant le transfert (choisissez un gros fichier) faites un `netstat`. Trouvez sur la liste des sockets la socket de données du transfert.

3.1.2 Transfert de fichier sans client FTP

Tentez de transférer un fichier d'un serveur FTP public sans utiliser de client FTP, seule la commande `telnet` est autorisée. Indice : vous allez avoir besoin du serveur FTP local (lisez attentivement le RFC 959).

3.2 Programmation système

La construction du client FTP comporte deux parties distinctes : la partie programmation système et la partie programmation réseau.

Pour implanter la communication entre les divers processus du serveur FTP, vous utiliserez une structure en mémoire partagée permettant de demander l'exécution d'une action et d'en recevoir le résultat. Cette structure `struct t_par` est décrite ci-dessous :

```
/** Types des actions **/

#define        TYPE_CNCT       0        /* Connexion a un serveur FTP       */
#define        TYPE_CHRE       1        /* Changement de repertoire courant */
#define        TYPE_LSRE       2        /* Liste de repertoire              */
#define        TYPE_RCFC       3        /* Reception de fichier             */

/** Codes d'erreur */

#define ERREUR_AUCUNE       0
#define ERREUR_DIVERS       1

/** Structure pour les actions et les reponses **/

struct t_act{
        int         type;               /* Type de l'action (cf. TYPE_XXXX) */
        char        nom1[MAX_NOM];      /* Nom du serveur FTP (en general)  */
        char        nom2[MAX_NOM];      /* Nom du fichier (en general)      */
        };

struct t_rep{
        int         erreur;             /* Code d'erreur (cf. ERREUR_XXXX) */
        char        nom[MAX_NOM];       /* Nom de fichier en retour        */
        };

union t_par{
        struct t_act        act;        /* Une action                      */
        struct t_rep        rep;        /* Le resultat de l'action         */
        };
```

De plus trois sémaphores gérent l'accès à cette mémoire. Le premier `SEM_EXCLUSION` assure qu'un seul processus peut accéder à la mémoire. Le second `SEM_ACTPRET` fait patienter le serveur d'actions jusqu'à ce que la structure `struct t_act` soit prête, le troisième sémaphore `SEM_REPRET` est le symétrique pour la structure de réponse `struct t_rep`.

3.2.1 Optimisation

Est-il possible de réduire le nombre de sémaphores utilisés ? Quel est l'inconvénient du sémaphore d'exclusion mutuelle `SEM_EXCLUSION` ? Comment pourrait-on s'en passer ?

3.2.2 Création des IPCs

Écrivez une fonction `void init_ipc(void);` pour créer la mémoire partagée et les sémaphores, respectivement avec les clefs `CLEF` et `CLEF`+1 (vous pouvez prendre, par exemple, 0x214356 comme valeur pour CLEF). Faites attention à l'initialisation des sémaphores. Cette fonction fait partie du fichier du serveur d'action `actsrv.c`.

3.2.3 Bibliothèque de communication

Vous allez programmer quelques fonctions pour la communication inter-processus et en faire une bibliothèque `comlib.a`. Les fonctions à programmer sont décrites ci-dessous.

- `int P(int sem,int num); int V(int sem,int num);`

  Ces fonctions implantent les actions P (puis-je) et V (vas-y) sur le sémaphore de numéro `num` dans le tableau de sémaphores `sem`.

- `void attacher_ipc(union t_par **parameter,int *sem);`

  Fonction de récupération de la mémoire partagée et des sémaphores. On suppose que la mémoire a été créée avec la clef `CLEF` et les sémaphores avec la clef `CLEF`+1. Cette fonction doit de plus attacher la mémoire à la variable `*parameter`.

- `void supprimer_ipc(int mem, int sem);`

  La fonction détruit la mémoire partagée et les sémaphores.

- `struct t_rep *envoi_act(union t_par *mem,int sem,struct t_com *action);`

  Cette fonction se charge d'envoyer `action` au serveur d'actions et de récupérer la réponse en utilisant la mémoire partagée `mem` et le tableau de sémaphores `sem`. Un pointeur sur la structure de réponse est retournée par la fonction.

3.3 Programmation réseau

La principale structure de données liée à la programmation réseau est la table des connexions décrite ci-dessous :

```
/** Nombre maximum de connexions FTP **/

#define MAX_CONNEXIONS  100

/** Etats des connexions **/

#define CON_ROMPUE      0
#define CON_ETABLIE     1
#define CON_IDENTIFIEE  2

/**** Structures ****/

/** Structure des connexions FTP en cours **/

struct t_con{
        struct in_addr  adresse;        /* Adresse du serveur FTP            */
        int             df;             /* Socket de controle FTP            */
        int             etat;           /* Etat de la connexion (cf CON_XXXX) */
        };
```

Une connexion est établie (`CON_ETABLIE`) dès que la socket de contrôle est créée avec succès, elle est identifiée (`CON_IDENTIFIE`) quand le serveur d'action s'est déclaré en tant qu'invité.

3.3.1 Une bibliothèque de fonctions réseau

Vous allez créer un fichier `scklib.c` contenant quelques fonctions réseau bien utiles. En voici la liste :

- `int adresse(char *hote,struct in_addr *adresse);`

  Cette fonction cherche l'adresse IP `adresse` d'une machine à partir de son nom `hote`. Elle retourne 0 si succès 1 sinon.

- `int cherche_port(char *nom);`

  Fonction retournant le numéro d'un port TCP connu par son nom `nom`.

- `int connecte_socket(char *hote,int port);`

  La fonction ouvre une socket et tente de la connecter au port `port` sur la machine `hote`. En cas d'échec elle retourne -1 sinon elle retourne le descripteur de la socket.

- `int init_socket(char *hote,int *port);`

  Cette fonction ouvre une socket en attente sur la machine locale avec le numéro de port `*port`. Si le port demandé est 0, le numéro de port assigné par le système est positionné dans `*port`. La fonction retourne le descripteur de la socket ou -1 en cas d'échec.

- `int ecoute_socket(int socket,char *nom);`

  Cette dernière fonction accepte une connexion sur la socket `socket` et écrit les données reçues dans le fichier `nom`. La fonction retourne -1 en cas de problème sinon 0.

3.3.2 Le serveur d'actions

Il faut maintenant compléter le fichier `actsrv.c` du serveur d'action. Les fonctions ci-dessous sont à écrire.

- `int cherche_connexion(char *hote);`

  La fonction recherche une connexion correspondant à `hote` dans la table des connexions et retourne son indice dans la table. Il faut retourner -1 si aucune connexion n'est trouvée.

- `int insere_connexion(char *hote);`

  Cette fonction se connecte au serveur FTP `hote` et insère les paramètres de cette connexion dans le tableau des connexions. L'état de la connexion passe à `CON_ETABLIE`. Si la connexion est déjà établie la fonction s'interrompt (la valeur de retour est 0 si tout se passe bien et -1 sinon).

- `int verifie_connexion(char *hote);`

  Il s'agit de vérifier qu'une connexion au serveur `hote` est établie et qu'elle fonctionne encore. La vérification se fait avec la fonction `teste_connexion` décrite plus bas. La signification de la valeur de retour est la même que celle de `cherche_connexion`.

- `int connecte_utilisateur(char *hote);`

  `int change_repertoire(char *hote,char *repertoire);`

  `int generique_fichier(char *hote,char *commande,char *fichier);`

Ce sont les fonctions implantant les actions décrites à la section 2.3. La dernière fonction implante les deux actions d'affichage de fichier et de liste. Pour ces fonctions vous avez intérêt à définir de petites sous-fonctions `écrire` et `lire` pour respectivement écrire sur la socket de contrôle et lire la réponse du serveur. Prenez garde qu'une réponse peut s'étaler sur plusieurs lignes, la dernière étant celle qui ne possède pas de tiret ('-') de continuation derrière le nombre.

- `int teste_connexion(int i);`

  Cette fonction teste à l'aide d'un `NOOP` que la connexion d'indice `i` dans la table de connexions est bien active (dans ce cas 0 est retourné sinon -1). En toute théorie il faut prévoir un temporisateur dans l'éventualité où le processus reste bloqué en écriture ou en lecture sur la socket de contrôle.

- `void traite_act(void);`

  Cette fonction utilise la mémoire partagée et les sémaphores pour recevoir l'action à effectuer et retourne la réponse de l'action (qui consiste généralement en le nom du fichier local dans lequel est stocké la liste ou le fichier rapatrié). La fonction fait, bien entendu, appel aux fonctions des actions concernées (voir ci-dessus).

- `main(int argc, char **argv);`

  Il faut ensuite appeler toutes ces fonctions dans le bon ordre. Il n'est pas interdit de dérouter des signaux pour, par exemple, supprimer proprement les IPC lors de la réception de `SIGTERM` ou `SIGINT`. Un autre perfectionnement consiste à se déconnecter des serveurs FTP contactés (à l'aide de la commande `QUIT`) sur réception de ces mêmes signaux `SIGTERM` et `SIGINT`.

3.3.3 Les `ftp`-commandes

Il ne vous reste plus qu'à implanter les commandes `ftpcd`, `ftpls` et `ftpcat`. Les deux dernières commandes étant très semblables, il est conseillé de ne faire qu'un seul exécutable générique qui se comportera différemment suivant le nom dans `argv[0]`. Il suffira ensuite de faire un lien symbolique pour obtenir l'autre commande.