

1 Le protocole FTP

Le protocole FTP veut dire « File Transfert Protocol » ou Protocole de transfert de Fichier. C'est le protocole de référence pour le transfert de fichiers. Il respecte une architecture client-serveur entre un client FTP et un serveur FTP.

FTP opère sur la couche 7 du modèle OSI (Application) et a besoin d'un protocole de transport fiable tel que TCP (s'utilise donc en mode connecté car si un paquet se perd cela engendre une donnée erronée chez le client ou le serveur) et il fonctionne par défaut sur le port 21 pour recevoir les commandes FTP et le port 20 pour recevoir les données.

La séquence de base de fonctionnement du protocole FTP est la suivante :

- établir une connexion
- envoyer son identifiant/nom d'utilisateur
- envoyer son mot de passe
- afficher le dossier courant
- changer de dossier courant
- récupérer une copie d'un fichier (download ou téléchargement)
- déposer une copie d'un fichier (upload ou téléversement)
- supprimer un fichier

FTP est dit **non sécurisé** car l'identifiant et mot de passe sont transmis en clair sur le réseau ainsi il est aujourd'hui plus judicieux d'utiliser SFTP (SSH File Transfer Protocol) ou FTP/S qui sont des dérivées de FTP chiffrant ces informations.

2 Implémentation et tests

FIGURE 1 – Capture wireshark

2.1 question 1

On crée un fichier à l'aide de la commande suivante pour qu'il possède une taille de 1024 mb (~1Gb) :

```
dd if=/dev/urandom of=target-file bs=1M count=1000
```

2.2 question 2, 3

Nous avons compilé le code à l'aide des commandes fournies. Nous avons vu que le code est fonctionnel. Effectivement, nous lançons, après la compilation, le serveur avec la commande :

```
./server
```

Le serveur se met à l'écoute.

Ensuite, après la compilation, nous lançons le client avec la commande :

```
./client
```

Nous voyons, comme prévu, que le texte du fichier appelé "file.txt" qui contient 1024 mb d'informations est bien transféré dans le fichier "file2.txt", après un certain temps.

Changeons le port du serveur et du client par un autre port (8080 par le port 20) :

```
int port = 20;
```

Après la compilation, testons la communication, la communication est réussie. Cependant, si nous changeons seulement le port du serveur par 20, nous voyons l'erreur suivante :

```
[–]Error in Connecting : Connection refused
```

Car le serveur n'écoute pas sur le bon port.

Notons que le code fourni comporte une erreur d'attribution des ports, à cause de cette erreur, les ports notés ici ne sont pas explicitement définis et donc ne sont pas attribués d'une manière souhaitée. En réalité nous n'attribuons pas le port 20 ou 8080 dans ce code. Cette erreur est corrigée à la question 5.

2.3 question 4

Nous lançons wireshark avec le paramètre `ip.addr == 127.0.0.1` pour trouver les échanges vu que nous travaillons en localhost dans notre programme client/serveur. Notons que la capture sera effectuée sur l'interface `NPF_Loopback`, nécessaire à la capture sur windows.


FIGURE 2 – Capture wireshark

2.4 question 5

On lance le serveur : `./server`

Le serveur crée la socket, l'attache, prévient le système auquel il appartient qu'il est prêt à accepter les demandes de connexion des clients (`listen`). Il se met en attente de demande de connexion. La socket de service n'est créée qu'après la demande de connexion, c'est elle qui permet d'avoir plusieurs connexions sur la même socket. Cependant, comme le port du serveur n'est pas connu du client, le triple handshake (le mode de fonctionnement TCP) ne pourra pas se réaliser.

La commande `netstat -ba` nous permet de voir quel programme (serveur) est en train d'écouter sur un port TCP, nous voyons :



Protocol	Source	Destination	Length	Info
TCP	127.0.0.1:5120	DESKTOP-HAVB8AT:0	60	LISTENING

Не удастся получить сведения о владельце

FIGURE 3 – Capture

C'est le seul serveur (shell linux sous windows 10) qui n'a pas pu être identifié et qui écoute sur le port 5120 à l'adresse 127.0.0.1. En effet, cette ligne disparaît si nous coupons le serveur. Nous allons changer l'adresse ip par 127.0.0.2 pour ne pas confondre cette adresse avec d'autres connexions (nous avons dans notre machine des serveurs locaux de chez Nvidia qui travaillent sur la même adresse ip).

TCP	127.0.0.2:5120	DESKTOP-HAVB8AT:0	LISTENING
Не удается получить сведения о владельце			

FIGURE 4 – Capture

Effectivement, nous voyons bien notre serveur en train d'attendre la connexion. Si nous ajoutons la ligne suivante dans le code :

```
printf("[+]Server port is :%d.\n",ntohs(server_addr.sin_port));
```

Nous nous apercevons que le port numéro 5120 est bien le vrai port qui a été attribué à notre serveur.

Ainsi pour faciliter les démarches, il faut corriger la façon dont les ports sont attribués dans ce programme serveur/client. Effectivement, nous allons corriger la ligne suivante :

```
server_addr.sin_port = port;
```

par :

```
server_addr.sin_port = htons(port);
```

La primitive htons permet une définition explicite des numéros de port, ainsi notre "port" comportera le numéro du port que nous souhaitons attribuer à notre programme.

Afin d'éviter des conflits et des erreurs d'attribution des ports sur notre machine, pour le reste du travail notre client/serveur vas travailler sur le port 2020.

2.5 question 6

Le client est actif, c'est donc le client qui fait la demande de connexion, nous avons le résultat suivant :

No.	Time	Source	Destination	Protocol	Length	Info
4	2.377114	127.0.0.1	127.0.0.2	TCP	56	50770 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
5	2.377127	127.0.0.2	127.0.0.1	TCP	44	8080 → 50770 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

> Frame 4: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{Loopback}, id 0	
> Null/Loopback	
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.2	
> Transmission Control Protocol, Src Port: 50770, Dst Port: 8080, Seq: 0, Len: 0	

FIGURE 5 – Capture

Le serveur ne crée pas de socket de service car il ne reçoit aucune connexion sur son port. Le client sur le port 50770 envoie un premier paquet *SYN* mais aucun serveur n'écoute sur le port 8080 qu'il sollicite. Ce paquet TCP inattendu (unexpected TCP packet) arrive à un autre hôte, l'hôte répond donc par un *reset packet* au travers de la même connexion. Cela indique que le port sollicité (8080) est fermé. Le paquet *ACK* retourné signifie simplement que l'hôte a bien reçu la demande de *SYN*.

Pour information : comme notre client n'utilise pas de primitive "blind" mais appelle une primitive "connect", la socket locale n'est pas attachée à une adresse, le système fera alors un attachement sur un port quelconque. Dans notre cas, c'est le port 50770.

2.6 question 7

Un processus Daemon est un processus qui s'exécute en arrière-plan et n'a donc pas de terminal de contrôle.

2.7 question 8

```
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ gcc server_daemon.c -o sd
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ time ./sd

real    0m0.003s
user    0m0.002s
sys     0m0.001s
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ ./client
[+]Server socket created.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ ./client
[+]Server socket created.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ ./client
[+]Server socket created.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$
```

On utilise la commande time afin de mesurer le temps d'exécution du programme lorsque l'on lance un client et le serveur, cela nous permettra de faire des comparaisons plus tard. Essayez d'ajouter au fur et à mesure du coup avec les captures que j'avais fait

2.8 question 9

On remarque que l'on a pas besoin de fork pour traiter plusieurs clients, on est pas dans le cas d'un serveur parallèle. Dans notre cas on les traite un à un grâce à une liste d'attente pour gérer les connexions des clients.

```
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
[+]File data send successfully.
[+]Disconnected from the server.
[+]Server socket created.
[+]Server socket created.
[+]Connected to server.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
[+]File data send successfully.
[+]Disconnected from the server.
[+]Server socket created.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
  PID  PGID  SID TTY      TIME CMD
  2327  2327  2327 pts/0    00:00:00 zsh
  2368  2368  2327 pts/0    00:00:00  bash
  3250  3250  2327 pts/0    00:00:00    bash
  3271  3250  2327 pts/0    00:00:00      ps
  2357  2355  2327 pts/0    00:00:00  zsh
  2356  2355  2327 pts/0    00:00:00  zsh
  2348  2347  2327 pts/0    00:00:00  zsh
  2359  2347  2327 pts/0    00:00:00 gitstatusd-linu
```

2.9 question 10

Par la suite nous avons complété le serveur pour qu'il devienne un serveur parallèle et puisse traiter plusieurs clients simultanément

2.10 question 11

On remarque que les clients peuvent bien se connecter au serveur en simultan .

```
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ l
client* file2.txt script_q12.sh* server* server_parallele.c
client.c file.txt script_q9.sh* server_daemon.c
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ gcc server_parallele.c -o sp
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ ./sp
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ ./client
[+]Server socket created.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ ./client
[+]Server socket created.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$ ./client
[+]Server socket created.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
eolienne@Eolienne:~/Documents/L3_S5/systeme/DM-2021/codes$
```

2.11 question 12

Comme pr c demment nous allons chron m tr  l'op ration mais cette fois-ci avec 3 clients sur le serveur parall le.

```
[+]Server socket created.
[+]Connected to server.
[+]Server socket created.
[+]File data send successfully.
[+]Disconnected from the server.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
[+]Server socket created.
[+]Connected to server.
[+]File data send successfully.
[+]Disconnected from the server.
  PID  PGID  SID TTY      TIME CMD
  2327  2327  2327 pts/0    00:00:00 zsh
  2368  2368  2327 pts/0    00:00:00 bash
  4657  4657  2327 pts/0    00:00:00 bash
  4675  4657  2327 pts/0    00:00:00 ps
  2357  2355  2327 pts/0    00:00:00 zsh
  2356  2355  2327 pts/0    00:00:00 zsh
  2348  2347  2327 pts/0    00:00:00 zsh
  2359  2347  2327 pts/0    00:00:00 gitstatusd-linu
real    0m0,139s
user    0m0,110s
sys     0m0,032s
```

2.12 question 13

Logiquement les temps doivent  tre proche l'un de l'autre voir similaire car le serveur parall le communique avec les 3 clients en m me temps donc ne devrait pas mettre plus de temps. Dans notre cas les chron m tres sont  loign s, nous avons remarqu  qu'en mode daemon le fichier se cr e mais ne r  crit pas ce qu'il y'a dedans ce qui fausse le r sultat. Cependant, on a observer avec wireshark que les paquets sont bien envoy s.

[illegible]

2.13 question 14

On en conclu que le serveur parallele va mettre moins de temps à traiter plusieurs client mais il utilisera bien plus de ressources car il va créer un fils pour chaque client.

3 Conclusion

Des manipulations précédentes on retiendra que FTP est :

D'un point de vue réseau un protocole de communication de niveau 7 qui nécessite un protocole de gestion de la connexion fiable TCP entre un ou plusieurs clients et un serveur pour pouvoir échanger des données. Ces échanges se passent au travers des ports 20 et 21 pour les données et la synchronisation des messages FTP.

D'un point de vue système, en FTP le client et le serveur sont des processus et lors de l'établissement de la connexion le serveur crée une socket d'écoute sur un port spécifique, l'attache (bind), prévient le système auquel il appartient qu'il est prêt à accepter les demandes de connexion des clients (listen). Ensuite se met en attente de demande de connexion. Lorsqu'un client tente de se connecter au serveur, ce dernier crée une autre socket sur un port différent (au travers d'un fork) qui lui sera dédiée. C'est ainsi qu'un serveur peut servir plusieurs clients simultanément.

4 Biblio/webographie

- Cours de Systeme, Pascal Fougeray, 2021
- https://fr.wikipedia.org/wiki/File_Transfer_Protocol
- <https://culture-informatique.net/cest-quoi-un-serveur-ftp/>