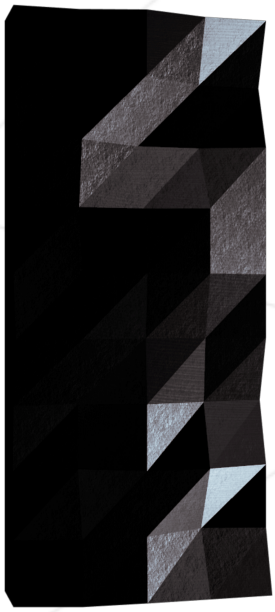

UNIX and network project

ft_p

42 staff staff@42.fr

Summary: This project is about implementing a client and a server that allow the transfer of files within a TCP/IP network.

Contents

Chapter I

Foreword

Here is what the Internet has to say about Truffade:

Truffade is a French potato and cheese dish. Considered a specialty in Auvergne, France, this dish is often served as a side in many restaurants and may be found as a snack food at local markets. Simple to make, the only required ingredients in truffade are potatoes and a mild cheese.

Goose or duck fat is traditionally used to sauté the dish, though butter is often used instead. Onions and garlic are generally included in truffade as well, and bacon may also be added. As with many dishes, salt and pepper are added as needed. Other spices may be occasionally included.

The cheese used in truffade is always mild. Cheddar or Gruyère, a type of yellow Swiss cheese, is the usual addition, though Cantal is suggested too. Tomme cheese, however, is the Auvergne region's preferred choice.

The potatoes are usually peeled, sliced, and then parboiled. Parboiling is a technique which partially cooks the food before it is added to a dish to cook fully. The onions are chopped and fried in the butter or fat. Usually garlic and any other seasoning are also added at this time.

Bacon may included in the dish as well. When bacon is used, it is generally chopped and fried in oil. Some recipes, however, suggest using the bacon grease as a substitute for the goose fat.

Once the onions, garlic, and bacon, if it is used, are fried, the potatoes are added. If the potatoes have not been parboiled, the dish is then covered while the potatoes cook. The cheese is only included after the potatoes have been browned. Otherwise, the cheese, and sometimes butter, is added directly after the potatoes have been stirred in. The dish is continually stirred while the cheese melts.

Although cheese is almost always the last ingredient, some recipes will add the garlic after the potatoes have cooked, and salt and pepper are usually added just before the cheese as well. Once the cheese has melted, the truffade is covered and cooked until the potatoes are browned. If the potatoes have been browned prior to the cheese addition,

the truffade only cooks long enough to melt the cheese. Then, the whole thing can be flipped onto a plate. Truffades are often served as a side to steak, but may accompany almost any dish.

There are few noticeable variations to truffades. Usually, the difference in recipes are slight, such as frying in goose fat versus butter. In the Dauphine region, however, tomatoes are included instead of the cheese.

Figure I.1: Truffade à la fourme d'Ambert

Chapter II

General Instructions

- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.
- The server's binary must be named `server`.
- The client's binary must be named `client`.
- A `Makefile` must compile both binaries and must contain the following rules: client, serveur, all, clean, fclean, re. It must recompile and re-link the programs only if necessary.
- Your project must be written in C in accordance with the Norm. Only norminette is authoritative.
- You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- You'll have to submit a file called `author` containing your usernames followed by a '\n' at the root of your repository.

```
$>cat -e author
xlogin$
$>
```

- Within the mandatory part, you are allowed to use the following functions:
  - socket(2), open(2), close(2), setsockopt(2), getsockname(2)
  - getprotobyname(3), gethostbyname(3), getaddrinfo(3)
  - bind(2), connect(2), listen(2), accept(2)
  - htons(3), htonl(3), ntohs(3), ntohl(3)
  - inet_addr(3), inet_ntoa(3)
  - send(2), recv(2), execv(2), execl(2), dup2(2), wait4(2)
  - fork(2), getcwd(3), exit(3), printf(3), signal(3)

- mmap(2), munmap(2), lseek(2), fstat(2)
- opendir(3), readdir(3), closedir(3)
- chdir(2), mkdir(2), unlink(2)
- The authorized functions within your libft (read(2), write(2), malloc(3), free(3), etc. for exemple ;-))
- select(2), FD_CLR, FD_COPY, FD_ISSET, FD_SET, FD_ZERO but only if it is to do something accurate!

- You are allowed to use other functions to complete the bonus part as long as their use is justified during your defence. Be smart!

- You can ask your questions on the forum, on slack...

Chapter III

Mandatory part

This project is about creating a client and a FTP server (File Transfer Protocol) that allows you to send and receive files between one or many clients and the server.

You are however free to choose the protocol you will use (you are not obligated to respect the RFC defining FTP, you can invent your own protocol of file transfers.) No matter your choice, you must however absolutely obtain a coherence between your client and your server. They must communicate properly with each other.

The communication between the server and the client will be in TCP/IP (v4).

Chapter IV

Server

Usage:

```
\$> ./serveur port
```

Where “port” means the server’s port number.

The server must support multiple simultaneous clients through a fork.

If you are very comfortable with select(2), you can use it. However you will have a chance to use it with the irc project, so use this project to learn how to create a server that forks.

If you are using select(2), do things correctly till the end:

- Select(2) in reading + writing (you don’t understand? Use fork(2))
- Server absolutely not blocking (you don’t understand? Use fork(2)).

If your server is badly designed because you wanted to use select(2) instead of fork(2), you will lose a lot of points in defense.

Chapter V

Client

Usage:

```
\$> ./client server port
```

Where “server” is the name of the hosting machine on which your server is, and “port” the port number.

The client must understand the following commands:

- ls : list in the current server’s directory.
- cd : change the current server’s directory.
- get _file_ : download the file _file_ from the server to the client.
- put _file_ : upload the file _file_ from the client to the server
- pwd : display the path of the current server’s directory.
- quit : cuts the connection + exit the program

and meet the following requirements:

- A specific prompts to the client (to distinguish it from Shell).
- Impossibility to lower to a level inferior to the server’s executable directory (unless a parameter specified to the server indicates another start directory).
- Display on the client of SUCCESS or ERROR messages + explanation after each request.

Chapter VI

Bonus part

We will look at your bonuses if and only if your mandatory part is EXCELLENT. This means that your must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. If that's not the case, your bonuses will be totally IGNORED.

Here are couple of bonus ideas:

- lcd, lpwd et lls: these functions concern the "local" filesystem and not the server.
- mget et mput: like get and put but "multiple", can contain some "*"
- login/password management
- rights control
- possibility to specify a different basic directory for each login
- conversion of '\n' en '\r\n' (unix<->windows) of files ("bin" and "asc" modes: binary = no conversion, ascii = conversion on the transferred file)
- prompt
- respect of RFC (standard 9 or rfc 959)
- IPv6 support
- completion during a get
- complétion during a put

Good luck everyone!