TZ20 - Rapport

PINARD Maxime - LAZARE Lucas

A2015

Sommaire

I. Introduction

Et si nous programmions nous même un serveur et son client, en partant de rien, plutôt que d'utiliser un programme préexistant ? C'est la question que nous nous sommes posés lorsque nous avons décidés de commencer ce projet. En effet, bien qu'il existe déjà de nombreux programmes pouvant faire cela — comme Apache, par exemple —, nous voulions comprendre la façon dont ces programmes fonctionnent. S'il est vrai que nous aurions pu simplement lire les sources d'Apache, nous avons pensés qu'il serait plus instructif de le refaire nous même. Ce faisant, nous devions décider de :

- Comment organiser les fichiers sur le serveur.
- Quelles sont les tâches respectives du client et du serveur.
- Comment ceux-ci communiquerons-t-ils.
- Quelles sont les autorisations du client.

Afin de répondre à ses questions, nous avons longuement réfléchi, ces questions étant importantes. L'organisation des fichiers, par exemple, règle pour beaucoup la façon dont le serveur fonctionne

II. Description des objectifs et énoncés des problèmes

L'objectif du projet TSiD est de coder un serveur et un client fonctionnels en ligne de commande, cela en utilisant la bibliothèque réseau SFML pour communiquer par internet. Les fonctionnalités seront les suivantes :

- L'accès au serveur est restreint aux membres qui possèdent un compte
- Chaque membre a accès à un dossier publique ainsi qu'a un dossier privé. Seul la personne a qui appartient le dossier privé peut y accéder
- Lorsque le client liste les fichiers d'un dossier, la date de création ainsi que le nom du créateur de chaque fichier lui sont aussi indiqués
- Un utilisateur, après avoir uploadé un fichier dans l'espace publique peut le supprimer dans les 24 heures
- Le serveur enregistre des logs de connexion
- Les utilisateurs peuvent changer de mot de passe
- Un utilisateur peut ajouter une description a un fichier qu'il a uploadé
- Un utilisateur peut créer un compte pour quelqu'un (c'est le seul moyen de créer un compte)
- Le serveur est configurable (ex : autoriser ou non aux utilisateurs l'accès a leur dossier privé)

Aussi, plus tard peut être, le serveur ne devra pas enregistrer les mots de passe des utilisateurs mais un hash de ceux-ci. Les deux programmes (client et serveur) doivent être disponibles sous les deux systèmes d'exploitation : Windows et Linux Pour implémenter ces fonctionnalités nous auront à résoudre les problèmes suivants :

- Comment communiqueront le client et le serveur ? Comment traiter les commandes envoyées par le client ?
- Comment traiter les accès aux dossier ainsi que les restrictions d'accès aux dossier ?
- Comment récupérer la liste des dossiers et fichiers présents dans un dossier ?
- Comment stocker des informations à propos des dossiers et fichiers uploadés ?
- Comment afficher des informations de façon ergonomique/lisible (avec des couleurs) ?
- Comment envoyer des mails via un programme ?
- Comment créer, gérer et utiliser la configuration du serveur ?
- Comment permettre au client de télécharger / uploader un dossier complet ?

Nous devont résoudre chaque problème sur les deux systèmes d'exploitation (Linux & Windows)

III. Solutions choisies

1. Client - server : communication

Afin de permettre la communication entre le client et le serveur, nous avons dû mettre en place une *'grammaire standard'*. Celle-ci fonctionne de la façon suivante :

- Le client envoie : *'Je veux faire ça, ici'*
- Le serveur essaiera ensuite de remplir la commande du client. Si cela est possible, il répondra au client *'Dáccord, fait le'*, ou renverra directement la réponse attendue. Sinon, il renverra la réponse apropriée (action non autorisée, une erreur est survenue, ...)
- Si le client doit effectuer une action supplémentaire (ie : téléverser un fichier), il le fera.

For the communication at its low level, which is sending and receiving a variable, we will use **SFML/Network** library that provides the functions required to send a variable across the internet, these are : **sf::TcpSocket::send(sf::Packet& packet)** and **sf::TcpSocket::receive(sf::Packet& packet)**, where a packet is an iostream. Here is a pseudo-code example : client wants to upload a file

Pseudo-code, client :

```
function uploadAFile( remote_working_directory: string,
   name_of_file_to_upload: string ): boolean
|
|   server_answer: integer
|   file: table of characters
|
|   sendToServer( upload ) //where 'upload' is associated to a
   command code (integer)
|   sendToServer( remote_working_directory )
|   receiveFromServer( server_answer )
|
```

```
    if isPositive( server_answer ) = true then
    |
    |    file <- readFile( name_of_file_to_upload )
    |    sendToServer( name_of_file_to_upload )
    |    sendToServer( file )
    |    return true
    |
    else
    |    return false
    endif
|
done
```

Pseudo-code, server :

```
procedure mainLoop()
|
|    directory: table of characters
|    command: integer
|
|    while true do
|    |
|    |    receiveFromClient( command )
|    |    receiveFromClient( directory )
|    |
|    |    switch command
|    |    |    .
|    |    |    .
|    |    |    .
|    |    |    case upload: retrieveAFile( directory )
|    |    |    .
|    |    |    .
|    |    |    .
|    |    end
|    done
done
```

```
procedure retrieveAFile( directory: table of characters )
|
|    file: table of characters
|    file_name: table of characters
|
|    if ClientIsAllowedToUploadThere( directory ) = true then
|    |
|    |    sendToClient( posivite_answer ) //where 'positive_answer' is
    associated to an answer code (integer)
|    |    receiveFromClient( file_name )
|    |    receiveFromClient( file )
|    |    write( file_name, file )
|    |
```

```
13 |    else
14 |    |    sendToClient( negative_answer ) //where 'negative_answer' is
       associated to an answer code (integer)
15 |    endif
16 done
```

2. Server : Restriction d'accès

note : *'./'* is the current folder, *'../'* is the parent's folder.

What we basically want to do here is to avoid people to access files that are not in the server itself, but that might be, for example, at the root of the filesystem. We also want to avoid the access by a member of another member's private folder.
This is easily done by analysing the path sent by the client.

For the first side of the part, we will assume that the client used by the member is the client we programmed and not a custom one, or that the member's client will have a behaviour similar to our own in term of path sending. In particular, our client is never supposed to input a *'../'* or a *'/..'*. As such, we just look for these sub-strings in the path, and, if we find them, we consider the client's tentative to be prohibited.

For the second part, we will use the server's folders architecture, wich is the following, the downloadable files (given foo and bar to be members) :

```
server execution folder/
                        - Public/
                                  - SomeFolders/
                                  - SomeFiles.ext
                        - Private/
                                  - foo/
                                        - SomePrivateFolders/
                                        - SomePrivateFiles.ext

                                  - bar/
                                        - SomePrivateFolders/
                                        - SomePrivateFiles.ext
```

File listing and client's pathing behaviour :

- The client starts at *'/'*, but this is interpreted by the server by *'/Public'*
- If the client wants to list files & folders in *'/Public'*, he sees all stuff in *'./Public'* plus an extra folder, wich is *'Private/'*.
- If the client wants to access *'./Public/Private'*, the server should reroot him to *'./Private/client_id'* silently (the client's display will simply shows it is in *'/Private'* ; remember *'/Public'* is *'/'*)

Given this, we will watch for the server's code, as there is not and should not be any restrictions checking from the client's side. We will also need to modify the main procedure given previously.

Pseudo-code, sever's restrictions (note that client_id is known) :

```
procedure main()
|
|   directory: table of characters
|   command: integer
|
|   while true do
|   |
|   |   recieveFromClient( command )
|   |   receiveFromClient( directory )
|   |
|   |   if formatPath( directory, client_id ) = true then
|   |   |
|   |   |   switch command
|   |   |   |   .
|   |   |   |   .
|   |   |   |   .
|   |   |   |   .
|   |   |   end
|   |   |
|   |   else
|   |   |   sendToClient( prohibited ) //where 'prohibited' is
    associated to an answer code (integer)
|   |   endif
|   done
done
```

```
function formatPath( directory: table of characters, client_id:
   table_of_characters ): boolean
|
|   if find( directory, "/../" ) = true then
|   //if you can find "/../" in directory
|   |   return false
|   endif
|
|   if endsBy( directory, "/.." ) = true then
|   //if directory ends by "/.."
|   |   return false
|   endif
|
|   if startsBy( directory, "/Private") = true then
|   //if directory starts by "/Private"
|   |   insert( directory, 9, "/"+client_id ) //inserts /client_id
    right after /Private
|   else
|   |   directory <- "/Public" + directory
```

```
18 |    endif
19 |    directory <- "." + directory //The directory is not from the
       disk's root, but from the server execution folder
20 |    return true
21 |
22 done
```

3. Client : Affichage

A readable display is required for the client so that the member may understand its output. For instance, if we make a really simple display like this :

```
Pictures/ sample.mp3 sample.mp4 Movies/ Documents/ Private/
```

This is, indeed, easy to implement. However, that's quite hard to read. Instead, we decided to :

- Display Files / Folders line by line
- Display firstly folders in blue, and then files in green, separated with a blank line
- Sort folders and files in alphabetical order
- Add a column with creation date and another with creator's id

the ouput displays looks like this :

```
Name          Creat. Date  Creator
Documents/
Movies/
Pictures/
Private/
sample.mp3    Fri 06/11/15 foo
sample.mp4    Sat 07/11/15 bar
```

We also decided to make a *'pacman style'* percentage display for downloads/uploads. For this display, we have to adapt pacman's line length to the window's length. Rules are the following :

- if there is less than 3 characters available, *'—'* are displayed.
- if there is between 11 and 34 characters available, the name and the already transfered byte's number is displayed.
- if there is between 35 and 45 characters available, the name and the number of transfered bytes over the number of bytes transfered is displayed.
- if there is more than 45 characters available, the above is display followed by pacman style's percentage display, which takes at most 1/3 of the total available characters.

Color being system dependent, the function is programmed within guards

4. Server : Stockage des informations sur les fichiers

The objective is to store basic informations about uploaded files, such as the upload date and the creator. To do so, we will use the following architecture :

```
server execution folder/
                        - Public/
                                  - SomeFolders/
                                  - SomeFiles.ext
                        - Private/
                                  - foo/
                                        - SomePrivateFolders/
                                        - SomePrivateFiles.ext

                        - FilesData/
                                    - Public/
                                              - SomeFolders/
                                              - .SomeFolders
                                              - .SomeFiles.ext
                                    - Private/
                                              - foo/
                                                    - SomePrivateFolders/
                                                    - .SomePrivateFolders
                                                    - SomePrivateFiles.ext
                                                    - .SomePrivateFiles.ext
```

Where *.SomeFolders*, *.SomeFiles.ext*, ... contains informations about *SomeFolders/*, *SomeFiles.ext*, ... The reason why a dot is added is so that we can write description about a folder and the folder itself in the same directory. Note that the creation of directories starting by a dot should not be allowed.

So all we need to do when foo uploads *'file'* in *'directory/'* (where directory is already formated as seen above) is :

- Write *'file'* in *'directory/'*
- Write the date and foo in *./FilesData/directory/.file*

Here is a pseudo-code example, modifying the retrieveAFile procedure : (No changes to the client)

Pseudo-code, server :

```
1 procedure retrieveAFile( directory: table of characters, client_id:
    table of characters )
2 |
3 |   file: table of characters
4 |   file_name: table of characters
5 |
6 |   if ClientIsAllowedToUploadThere( directory ) = true then
7 |   |
8 |   |   sendToClient( posivite_answer ) //where 'positive_answer' is
    associated to an answer code (integer)
```

```
|    |    receiveFromClient( file_name )
|    |    receiveFromClient( file )
|    |    write( file_name, file )
|    |    writeFileInformations( directory, file_name, client_id )
|    |
|    else
|    |    sendToClient( negative_answer ) //where 'negative_answer' is associated to an answer code (integer)
|    endif
done
```

```
procedure writeFileInformations( directory: table of characters, file_name: table of characters, client_id: table of characters )
|
|    date: table of characters
|
|    date <- retrieveDate()
|    makeDirectory( directory )
|    write( date + NEWLINE + client_id, "./FilesData" + directory + "." + file_name ) //writes the file's info in FilesData/directory/.filname
|
done
```

5. Client : upload / download récursif

In order to allow the client to dowload and upload a whole folder, we decided to use a recursive approach, working as the following :

1 Open the folder and see the first element it contains

2a If it is a folder, step back to 1 with this new folder, then step forward to 3

2b If it is a file, download/upload it, then step forward to 3

3 Step back to 2 with the next element of the foder

There is the recursive download pseudo-code (no changes to the server)

Pseudo-code, client :

```
function recursiveDownload( remote_working_directory: table of characters ): boolean
|
|    successful: boolean
|    server_answer: integer
|    i: integer
|    file_list: table of table of characters
|
|    sendToServer( listFiles )        //Where 'listFiles' is associated to a command code
```

```
9  |   sendToServer( remote_working_directory )
10 |   receiveFromServer( server_answer )
11 |
12 |   if server_answer = negative_answer then //Where 'negative_answer
   is associated to some answer codes
13 |   |   return false
14 |   endif
15 |
16 |   successful <- true
17 |
18 |   receiveFromServer( file_list )      //file_list now contains a
   list of the files and folders that are within
   'remote_working_directory'
19 |
20 |   for i in [1..file_list.size()]      //from 1 to the number of
   files/folder within 'file_list'
21 |   |
22 |   |   if endsBy( file_list[i], "/" ) then //if 'file_list[i]' ends
   by '/', it is a folder
23 |   |   |   recursiveDownload( remote_working_directory + '/' +
   file_list[i] )
24 |   |   |
25 |   |   else
26 |   |   |   success <- download( remote_working_directory + '/' +
   file_list[i] ) and success    //'success' is true if the download
   was successful AND if it was true before
27 |   |   endif
28 |   done
29 |
30 |   return success
31 |
32 done
```

6. Server : Configuration

We want to have a configuration for the server. There is a list of the configurable elements for now :

- Generate or not the server folders at next startup
- Create a new user or not at next startup
- Allow members to invite another member (basically : create a new account) or not
- Allow members to write in their private folders or not
- Allow members to download from their private folders or not

To do so, we decided to create an object Config, having the following private variables :

- user_creation_allowed
- private_folder_writing_allowed
- private_folder_reading_allowed

all booleans
This object will be in read-only.
The first two setting do not appear in the object, as they are used only at startup.
A pointer to this object will be passed to each client's thread.

7. Server, client : Message d'accueil

The objective is to allow the the server owner to write a user-adjustable welcome message for all users. In this way we wanted to implement some variables in the welcome message, so we used the syntax *'$[variable_name]'*, the available variables are :

- **user** the user name
- **date** the date formated *'dd/mm/yy'*
- **day** the day formated *'Mon, Tue, Wed, Thu...'*
- **hour** the hour formated *'hh :mm'*
- **color** to set the text color from the variable to the next color variable

where **color** can be blue, green, cyan, red, magenta, yellow, white. To put the *'$'* symbol simply put *'$$'*

Pseudo-code, server :

```
procedure mainLoop()
|
|    formatedWelcomeMessage( message, client_id )
|    SendToClient( message )
|
|    while ClientConnected() do
|    |    .
|    |    .
|    |    .
|    done
done
```

```
Procedure formatedWelcomeMessage(message: table of characters,
   client_id: table of characters)
|
|    read( "WelcomeMessage.txt", message )
|
|    foreach $[command] in message do
|    |
|    |    switch command
```

```
|   |   |
|   |   |   case user: remplace( $[command], client_id )
|   |   |
|   |   |   case date: remplace( $[command], getDate() )
|   |   |              //formated 'dd/mm/yy'
|   |   |
|   |   |   case day: remplace( $[command], getDay() )
|   |   |              //formated 'Mon, Tue, Wed, Thu...'
|   |   |
|   |   |   case hour: remplace( $[command], getHour() )
|   |   |              //formated 'hh:mm'
|   |   |
|   |   |   default: //let the client interpret $$ and $[color]
|   |   end
|   done
done
```

Pseudo-code, client :

```
Procedure main()
|
|   connectToServer()
|   InterpretWelcomeMessage()
|
|   while ConnectedToServer() do
|   |   .
|   |   .
|   |   .
|   done
done
```

```
Procedure InterpretWelcomeMessage()
|
|   ReceiveFromServer(message)
|
|   while( not_the_end_of_the_message )
|   |
|   |   Print( GetTextUntilSymbol(message, '$') )
|   |
|   |   switch( GetNextChar(message) ) //switch char after $
|   |   |
|   |   |   case '$': Print('$') //there is $$ in the message, put $
|   |   |
|   |   |   case '[': //there is $[color] in the message, change the
    color
|   |   |   |
|   |   |   |   color <- GetTextUntilChar(']') //color take the value
    in $[color]
|   |   |   |
|   |   |   |   if( IsAPrintableColor( color ) ) then
```

```
18 |   |   |   |   |
19 |   |   |   |   |   SetTextColor( color )
20 |   |   |   |   |
21 |   |   |   |   endif
22 |   |   |   |
23 |   |   |   break
24 |   |   end
25 |   done
26 done
```

IV. Conclusion

Ce projet fut très instructif et intéressant. La dernière version du programme pour l'UV de TZ20 et la V1 Ranitomeya reticulata, elle ne contient pas tout ce que nous aurions voulu implémenter mais les fonctionnalités les plus importantes ont été implémentées. Nous avons la volonté de continuer à travailler sur ce programme. La version actuelle peut être utilisée avec des personnes en lesquelles vous avez confiance, mais la sécurité n'ayant pas été testée nous ne recommandons pas son utilisation à plus grande échelle.

1. Améliorations possibles

Certaines fonctionnalités mériteraient d'être plus développées mais ne l'ont pas été par manque de temps, par exemple :

- **Suppression de fichiers**
  Les utilisateurs peuvent uploader des fichiers et dossier mais ils ne peuvent pas les supprimer. Nous avions pensé à un système de vote ou les utilisateurs auraient pu voter pour ou contre la suppression d'un fichier.

- **Mots de passe**
  Les mots de passe sont enregistrés tel-quel sur le serveur ce qui n'est pas assez sécurisé. Il serait préférable de sauvegarder des hash de ces derniers.

- **Mails**
  Nous aurions voulu implémenter un système de mails qui aurait pu être utilisé pour :
  - Inviter des nouveaux utilisateurs
  - Prévenir les membres lorsqu'une adresse IP inconnue essaye d'accèder a leur compte
  - Informer un membres lorsqu'un vote a été lancer pour la suppression d'un fichier qu'il a uploadé

- **Filtre IP**
  Dans le but d'aider les utilisateurs à sécuriser leur compte, nous voudrions ajouter une vérification de l'adresse IP à la connexion. Toute IP non autorisée se verra ainsi refuser la connexion au compte. Un mail pourra être envoyé à l'utilisateur pour ajouter l'IP à la liste des adresses autorisées. L'utilisateur aurait la possibilité de désactiver ce service.

- **Admin thread**
  There is no way to delete a user, a file, or to change the server configuration easily, which is not nice. The problem could be solved by adding an admin console, were you could enter some commands

- **Limit private folder space usage**
  The goal of this server is mainly to use the public folder, so it could be nice to set a limit to the private folder, let's say 10Gio, configurable via the server's configuration

2. Connaissances acquises

What did we learn while doing this project ?

- **Teamwork**
  We learned to work as a team, in a different way we did in others UV with a team presentation, such as LE03

- **Work organization**
  As we were working in autonomy, we had to organize us in order to meet the deadlines. We had to organize both the working order (what to do), and timing (when to do it). That may be very useful in our professional life.

- **GitHub**
  GitHub is a powerful tool of version gesture that helps developpers to work together on a project. This could help us as well in our active life, as we may work as developpers, and our company will probably use a similar tool

- **Strings manipulation**
  We learned a lot about strings [in C++], and that knowledge can easily be transfered most of others programming languages

- **Cmake**
  Finally, we learned about CMake, which is a powerful tool to generate MakeFiles.