

Présentation TZ20

Lazare Lucas - Pinard Maxime

UTBM

10 décembre 2015

1 Objectifs

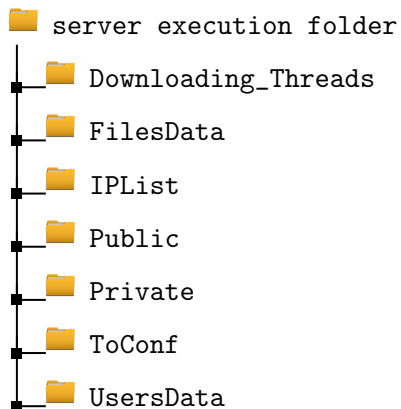
2 Problèmes

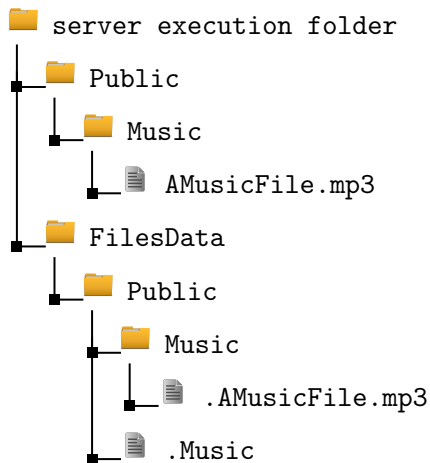
3 Solutions

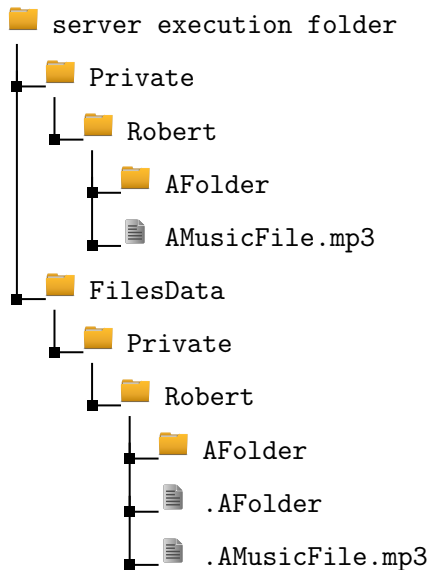
4 Exemple

- Dossiers privés et publique
- Changement de mot de passe
- Descriptions de fichiers
- ...

- Communication client - serveur
- Gérer les restrictions d'accès
- Afficher des informations de façon ergonomique/lisible
- ...







```

Terminal - organic-code@Ninjabo:~/run/media/organic-code/Shared
Connect as Foo@localhost:1234 ?[Y\n]
User passwd :

Connecting to the remote @ localhost:1234
Connected to server

Welcome to you Foo !

We're happy to see you back there !

Foo@TSiD / $ put Awesome/*
Upload is starting
Awesome/Archive.zip                               422 MiB 3583 MiB [--c 0 0 0 0 0 0 0 0 0 0]

```

```

Terminal - organic-code@Ninjabo:~/Server_folder

Copyright (C) 2015 Lucas Lazare and Maxime Pinard
Program under MIT License : <https://github.com/Organic-code/TSiD/blob/v1/LICENSE>
This is a free software : you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Maxime Pinard and Lucas Lazare

~ server started
[10:42] * Client found
[10:42] Foo - connected
[10:42] Foo -> welcome message send
[10:43] Foo : directory creation request
[10:43] * directory ../Public//Awesome created
[10:43] * directory ../FilesData/Public//Awesome created
[10:43] * file ../FilesData/Public//Awesome created
[10:43] Foo -> directory exist
[10:43] Foo - directory creation request successfully answered
[10:43] Foo : upload request
        -File: ../Public/Awesome/Archive.zip
        -File size: 3 GiB
[10:43] * file ../Public/Awesome/Archive.zip created
[10:43] * Start downloading ../Public/Awesome/Archive.zip from Foo
[10:43] Foo - [0%] of download

```



```
//error
client.packet.clear();
client.packet << UnknownIssue;
client.socket.send(client.packet);
tprint();
std::cout << client.name() << " -> There was an error [...]" << std::endl;

bool a_retrieveData(Client& client){

    unsigned int file_size;
    unsigned int bytes_per_packet;

    if( !(client.packet >> file_size >> bytes_per_packet) ){
        //error
        return false;
    }
    std::cout << "\t-File: " << client.path << std::endl;

    if( file_size == 0 ){
        //error
        return false;
    }
}
```

```
if(file_size < 1024 ){
    std::cout << "\t-File size: " << file_size << " B" << std::endl;
}

else if(file_size < 1024 * 1024 ){
    std::cout << "\t-File size: " << file_size/1024 << " KiB" << std::endl;
}

else if(file_size < 1024 * 1024 * 1024 ){
    std::cout << "\t-File size: " << file_size/(1024 * 1024)<< " MiB" <<
        std::endl;
}

else{
    std::cout << "\t-File size: " << file_size/(1024 * 1024 * 1024)<< "
        GiB" << std::endl;
}
```

```
client.packet.clear();

switch(createFile(client.path)){

    case AlreadyExist:
        client.packet << AlreadyExist;
        client.socket.send(client.packet);
        tprint();
        std::cout << client.name() << " -> File already exists" <<
            std::endl;
        return false;
        break;

    case UnknownIssue:
        //error
        return false;
        break;

    default:
        break;
}
```

```
std::ofstream output_file ( client.path.c_str(), std::ios::binary |
    std::ios::out );

unsigned int loop_number(file_size/bytes_per_packet);
char* input_data_array = new char[bytes_per_packet];
sf::Int8 input_data;
unsigned char percentage_count(0);

client.packet.clear();
client.packet << ServerReady;
client.socket.send(client.packet);

tprint();
setColors("light blue");
std::cout << "* Start downloading " << client.path << " from " <<
    client.name() << std::endl;
setColors("reset");
```

```
for( unsigned int i(0) ; i<loop_number ; ++i){

    client.packet.clear();

    if(client.socket.receive( client.packet ) == sf::Socket::Disconnected){
        //error
        output_file.close();
        removeFile(client.path);
        return false;
    }

    for( unsigned int j(0) ; j<bytes_per_packet ; ++j ){
        client.packet >> input_data;
        input_data_array[j]=static_cast<char>(input_data);
    }

    output_file.write( input_data_array, bytes_per_packet );

    if( static_cast<unsigned char>(100*i/loop_number) > percentage_count ){
        tprint();
        std::cout << client.name() << " - ";
        setColors("light blue");
        std::cout << "[" << static_cast<short>(percentage_count) << "%";
        setColors("reset");
        std::cout << " of download" << std::endl;
        percentage_count = static_cast<unsigned char>(percentage_count +
            25);
    }
}
```

```
file_size -= loop_number * bytes_per_packet;
if( file_size > 0 ){

    client.packet.clear();
    if(client.socket.receive( client.packet ) == sf::Socket::Disconnected){
        //error
        output_file.close();
        removeFile(client.path);
        return false;
    }

    for( unsigned int j(0) ; j < client.packet.getDataSize() ; ++j){
        client.packet >> input_data;
        output_file << static_cast<char>(input_data);
    }
}
```

```
output_file.close();

tprint();
std::cout << client.name() << " - ";
setColors("light blue");
std::cout << "[100%] ";
setColors("reset");
std::cout << " Transfer terminated successfully" << std::endl;
createInformationFile(client.path, client.name());
delete[] input_data_array;
return true;
}
```

```
char createInformationFile(std::string path, std::string user_name){  
  
    std::string info_path = "./FilesData" + path.substr(1, std::string::npos);  
    //add "./FilesData" at the begening  
  
    if( fileExist(info_path) ){  
        setColors("light red");  
        std::cout << "\t-The informations file already exist" << std::endl;  
        setColors("reset");  
        return AlreadyExist;  
    }  
  
    if(isFolder(path)){  
        createDirectory(info_path);  
    }  
}
```



```
info_path = info_path.insert(info_path.find_last_of("/") + 1, "."); //insert
    '.' before the filename
createFile(info_path);
std::ofstream file ( info_path.c_str(), std::ios::binary | std::ios::out );

if( file.fail() ){
    file.close();
    std::cout << "\t-";
    setColors("light red");
    std::cout << "Error writting in the informations file" << std::endl;
    setColors("reset");
    return UnknownIssue;
}

file << formattedTime() << std::endl;
file << user_name << std::endl;
file.close();
return Created;
}
```

```
bool sendData(sf::TcpSocket& server, std::string const& current_directory) {  
  
    std::ifstream input_file;  
    unsigned int file_size;  
  
    std::cin.ignore();  
    std::string filename;  
    std::getline(std::cin, filename);  
  
    if (filename.back() == '*') {  
        filename.pop_back();  
        filename.pop_back();  
        return recursiveUpload( server, current_directory, ".",  
                                filename );  
    }  
  
    if (!startUpload(input_file, file_size, server, current_directory,  
                    filename)) {  
  
        std::cout << "Could not send the file" << std::endl;  
        return false;  
    }  
    return uploadFile( server, input_file, file_size, filename );  
}
```

```

bool startUpload(istream& infile, unsigned int& file_size, TcpSocket& server,
string const& dir, string filename)

    std::string directory(dir);
    formatDir(directory);
    if (isFolder(filename)) {
        std::cout << "You are trying to upload a folder... (maybe you
            forgot to add * ?)" << std::endl;
        return false;
    }
    file_size = getFileLength(filename);
    infile.open(filename.c_str(), std::ios::binary | std::ios::in);
    if (file_size == 0 || infile.fail() ) {
        std::cout << "There was a problem reading the file : " <<
            filename << " (maybe that this file is empty ?)" <<
            std::endl;
        return false;
    }
    sf::Packet packet;
    packet << (directory+'/' +formatPath(filename)) << Upload << file_size
        << NB_BYTE_PER_PACKET;
    server.send(packet);
    packet.clear();

    int server_state;
    server.receive(packet);
    if (packet.getDataSize() > sizeof( int ) || !(packet >> server_state)){

        std::cout << "There was an error retrieving server state" <<
            std::endl;
        return false;
    }
    return interpretServerAns(static_cast<char>(server_state));

```

```
bool sendData(sf::TcpSocket& server, std::string const& current_directory) {  
  
    std::ifstream input_file;  
    unsigned int file_size;  
  
    std::cin.ignore();  
    std::string filename;  
    std::getline(std::cin, filename);  
  
    if (filename.back() == '*') {  
        filename.pop_back();  
        filename.pop_back();  
        return recursiveUpload( server, current_directory, ".",  
                                filename );  
    }  
  
    if (!startUpload(input_file, file_size, server, current_directory,  
                    filename)) {  
  
        std::cout << "Could not send the file" << std::endl;  
        return false;  
    }  
    return uploadFile( server, input_file, file_size, filename );  
}
```

```

bool uploadFile (sf::TcpSocket& server, std::ifstream& input_file, unsigned int
    file_size, std::string const& filename)

    unsigned int loop_number=file_size/NB_BYTE_PER_PACKET;
    char input_data_array[NB_BYTE_PER_PACKET];
    sf::Packet spacket;
    spacket.clear();

    std::cout << "Upload is starting" << std::endl;
    percentageDisplay(0, filename, file_size, 0);

    for (unsigned int i(0) ; i<loop_number ; ++i) {

        input_file.read(input_data_array, NB_BYTE_PER_PACKET);
        for (unsigned int j(0) ; j<NB_BYTE_PER_PACKET ; ++j)
            spacket << static_cast<sf::Int8>(input_data_array[j]);

        if (server.send(spacket) == sf::Socket::Disconnected) {
            std::cout << "Lost connection with server !" <<
                std::endl;
            return false;
        }

        spacket.clear();

        if (i%10 == 0)
            percentageDisplay( static_cast<unsigned
                char>(100*i/loop_number), filename, file_size,
                i*NB_BYTE_PER_PACKET );
    }

```

```

bool uploadFile (sf::TcpSocket& server, std::ifstream& input_file, unsigned int
    file_size, std::string const& filename)

    file_size -= loop_number * NB_BYTE_PER_PACKET;
    if (file_size > 0) {
        char* file_tail = new char[file_size];
        input_file.read( file_tail, file_size);
        for (unsigned int j(0) ; j< file_size ; ++j)
            spacket << static_cast<sf::Int8>(file_tail[j]);

        if (server.send(spacket) == sf::Socket::Disconnected) {

            std::cout << "Too bad. You almost done it but you were
                disconnected by server :(" << std::endl;
            delete file_tail;
            return false;
        }

        percentageDisplay( 100, filename, file_size + loop_number *
            NB_BYTE_PER_PACKET, file_size + loop_number *
            NB_BYTE_PER_PACKET );

        delete file_tail;
    }

    std::cout << std::endl << "Transfer terminated successfully" <<
        std::endl;
    input_file.close();
    return true;

```

Avez-vous des questions ?