FFT

“Freaky File Transfer”

Di Pietro Filippo

July 29, 2022

Chapter 1

Problem Analysis

FFT is a software that allows the transfer of files through a server and client and vice versa, using the UDP protocol.

The software supports these operations:

- ls: show files on the server
- get: download a file from the server
- put: upload a file to the server
- No authentication

Chapter 2

2.1 Design

The software follows an OOP design, these are the classes used by the software:

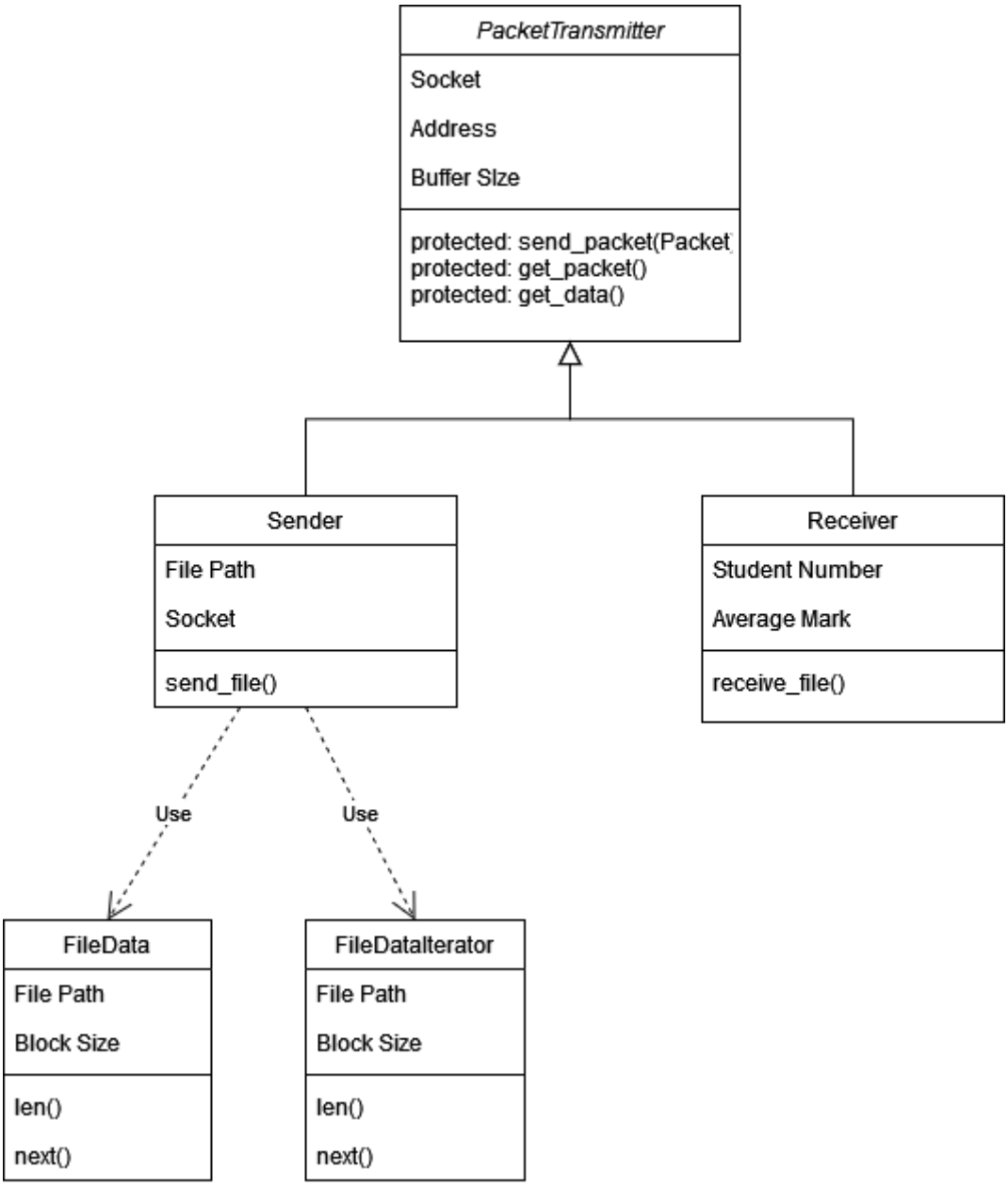
- Packet, this class model a packet of data (any kind of data), that is simple to send

- FileData, FileDataIterator, for instantiate an object of these classes need to provide a file path, the task of these objects is to split the provided file into a block with a determinate number of bytes per block. The difference between FileData and FileDataIterator is that the last one has a lazy approach, so every time the next is called on a FileDataIterator object the next block of data of the file is read, the first one instead on the construction of the object, all the content of the file is split into a list of blocks.

- PacketTransmitter is a class that model an object that can send Packet, but need to be defined the close method, which is abstract, it deals to close the socket, but strictly depends on the subclass so it's abstract.

- Sender is a class that model an object that can send a file, is created by the client when put command is executed, or server when the get command is executed. The Sender has a FileData (or FileDataIterator) object containing each block that need to be sent to the Reciver.

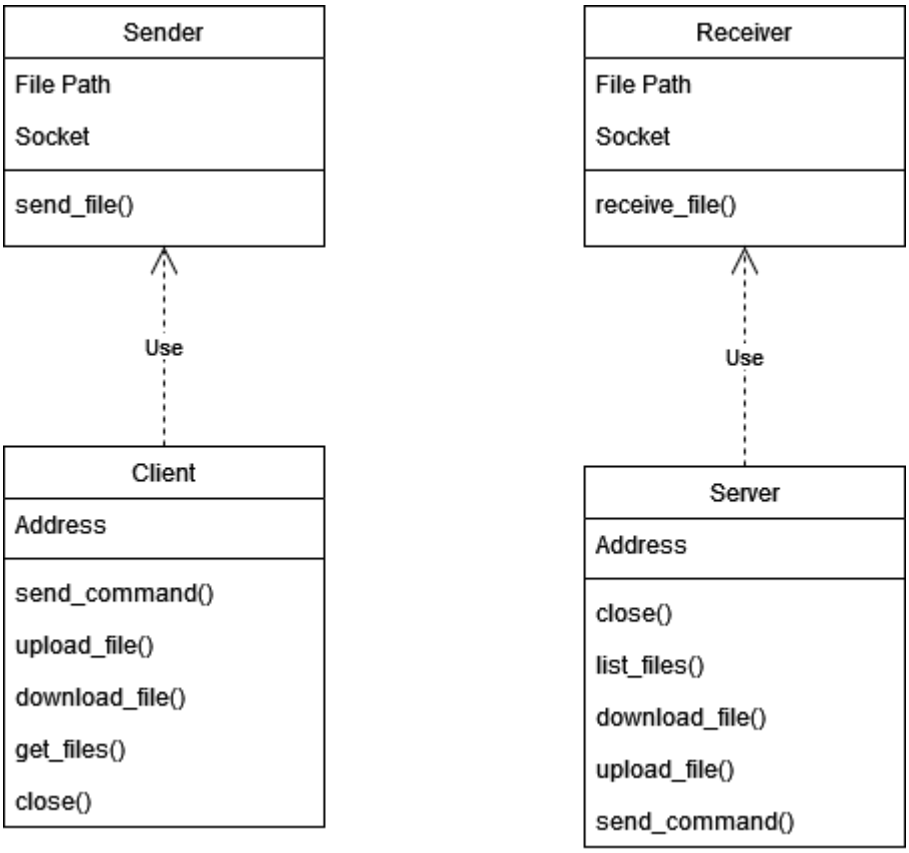
- Receiver is a class that model an object that can receive a file from a sender, is created by the client when put command is executed, or server when the get command is executed



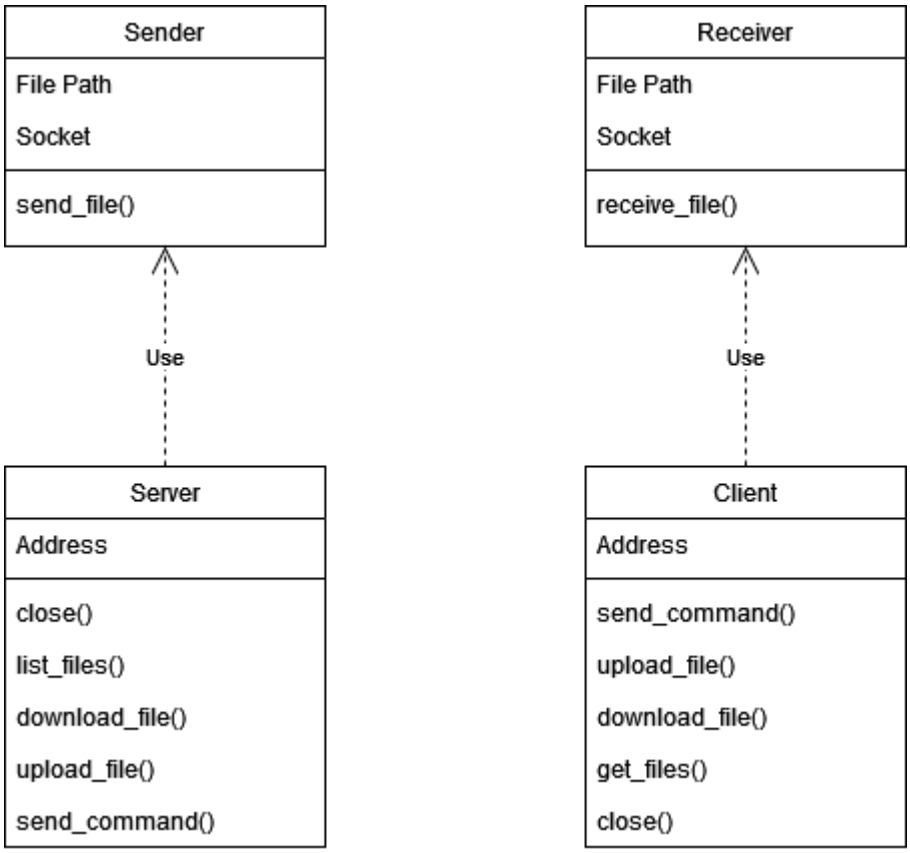
This figure explains the design of the classes

- Client is a class that model a client, so it supports all the previous commands, can send it to the server, and handle answer from it

- Server is a class that mode a server, so it supports all the previous commands, can send answer to the client.



This figure explains which objects instantiate server and client during a put (upload)



This figure explains which objects instantiate server and client during a get (download)

2.2 Protocols

2.2.1 Sender & Reciver

1. Sender) first sends “exts” if the file requested exists, otherwise send “doesn’t exts”, and raise an IOError

2. Receiver) if receive “exts” prepare to receive the entire file. Otherwise, if receive “doesn’t exts”, raise an IOError

3. Sender) Once the function ***send_file()*** is called, the sender sends the number of blocks

4. Reciver) Read the number of blocks

5. Sender) Start sending’s every block

6. Receiver) Receive the block and write it into a file

For each block sent by Sender:

1. Sender) send a block of file

2. Reciver) receive the block, checks the hash, if correct the receiver sends “next”, if not correct the receiver sends “re-send”, instead if the block is not received will be sent a “re-send” command, until the server response, with the new block

3. Sender) if “next” was received the next packet will be sent, otherwise resend the current packet

2.2.2 Server & Client

Server: one the server object is created it waits for a client command (ls, get, put)

Client: once the client object is created it waits for user input

ls:

1. Client) send ls to server
2. Server) respond with ACK
3. Client) handle ACK
4. Server) send files
5. Client) receive files, displays all files to the user

get:

1. Client) send get to the server
2. Server) receive the command and prepare to get the file name
3. Client) send file name
4. Server) receive the file name
5. Server) send ACK
6. Server) instantiate a Sender object
7. Client) instantiate a Receiver object

put:

1. Client) send put to the server
2. Server) receive the command and prepare to obtain the file name
3. Client) send file name
4. Server) receive the file name
5. Server) send ACK
6. Server) instantiate a Receiver object and prepare receiving
7. Client) instantiate a Sender object and start sending

2.3 Data integrity

For handling the problem of integrity of the data, for every Packet sent, will be sent a hash which check for correctness of the data filed. For checking the valid data will be used the classmethod ***by_json()***, which takes data and the hash, now evaluate the hash on the data checks if is identical to the hash provided then if is equal will be returned a Packet on the data provided.

2.4 Data delivery

For sending a Packet, the class need to be a subclass of PacketTransmitter. The ***send_packet()*** method encodes the data in the Packet into json, next will be encoded in bytes, so is able to send.

2.5 Large files

For Handling large files, the sender will use FileDataIterator, so will save large amount of memory

Chapter 3

User Guide

Run a client

```
python client.py
```

or

```
python3 client.py
```

it depends on your python versions

next when the arrow is displayed you can write the three commands mentioned before (ls, get, put), for put a get is optional to provide the filename after a space, if the file name is not provided will be asked after an hit of enter

Run a server

```
python server.py
```

or

```
python3 server.py
```

it depends on your python versions