package ca.polymtl.inf4410.tp1.client;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.PrintWriter;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.rmi.AccessException;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.HashMap;

import java.util.Map.Entry;

import java.util.Scanner;

import ca.polymtl.inf4410.tp1.shared.ServerInterface;

public class Client {

public static void main(String[] args) {

if(args.length > 0){

Client client = new Client();

client.run(args);

} else {

System.out.println("Please specify a command. Use \"help\" to list available commands.");

}

}

private ServerInterface localServerStub = null;

private Integer clientId;

public Client() {

super();

if (System.getSecurityManager() == null) {

System.setSecurityManager(new SecurityManager());

}

localServerStub = loadServerStub("127.0.0.1");

}

private void run(String [] args) {

File f = new File("clientInfo.xqt");

if(f.exists() && !f.isDirectory()) {

//On lit ce qu'est l'identifiant du client (clientId)

try {

Scanner sc = new Scanner(f);

clientId = new Integer(sc.nextInt());

sc.close();

} catch (Exception e) {

// S'il y a une erreur lors de la lecture, on en creer un nouveau

// This should never happen...

createClientID();

}

} else {

createClientID();

}

try{

// exécution des commandes disponibles

switch(args[0].toLowerCase()){

case "help":

System.out.print("List of available commands for the client: \n\n"

+ "help \n"

+ "create filename \n"

+ "list \n"

+ "syncLocalDir \n"

+ "get filename \n"

+ "lock filename \n"

+ "push filename \n\n");

break;

case "create":

if(args.length >= 2){

if(localServerStub.create(args[1])){

System.out.println("File " + args[1] + " created on server.");

} else {

System.out.println("File " + args[1] + " not created: Already exists on server.");

}

} else {

System.out.println("Please specify filename to create. \n"

+ "Syntax: create filename\n\n");

}

break;

case "list":

HashMap<String,Integer> data = localServerStub.list();

System.out.println("List of files on server:");

for (Entry<String, Integer> entry : data.entrySet()) {

String key = entry.getKey();

Integer value = entry.getValue();

System.out.print("\* " + key + "\t");

if(value == null){

System.out.print("unlocked\n");

} else {

System.out.print("locked by client " + value.intValue() + "\n");

}

}

int nbFiles = data.size();

System.out.print(nbFiles + " file(s)\n");

break;

case "synclocaldir":

HashMap<String, byte[]> fileData = localServerStub.syncLocalDir();

for (Entry<String, byte[]> entry : fileData.entrySet()) {

String key = entry.getKey();

byte[] value = entry.getValue();

File oldFile = new File(key);

FileOutputStream fStream = new FileOutputStream(oldFile, false);

if(value != null){

fStream.write(value);

}

fStream.close();

}

System.out.println("Local directory updated.");

break;

case "get":

if(args.length >= 2){

File fileToGet = new File(args[1]);

byte [] checksum = null;

if(fileToGet.exists() && !fileToGet.isDirectory()) {

// Le fichier existe déjà localement

byte[] bytes = Files.readAllBytes(Paths.get(args[1]));

checksum = getChecksum(bytes);

}

byte[] newData = localServerStub.get(args[1], checksum);

if(newData == null){

// Soit le checksum est égal donc le fichier est déjà à jour, soit il n'existe pas sur le serveur

System.out.println("File not updated - Already up to date or not present on server. Use \"list\" to confirm.");

} else {

File oldFile = new File(args[1]);

FileOutputStream fStream = new FileOutputStream(oldFile, false);

fStream.write(newData);

fStream.close();

System.out.println("File " + args[1] + " updated.");

}

} else {

System.out.println("Please specify filename to get from server. \n"

+ "Syntax: get filename\n\n");

}

break;

case "lock":

if(args.length >= 2){

File fileToGet = new File(args[1]);

byte [] checksum = null;

if(fileToGet.exists() && !fileToGet.isDirectory()) {

//Le fichier existe déjà localement

byte[] bytes = Files.readAllBytes(Paths.get(args[1]));

checksum = getChecksum(bytes);

}

HashMap<Boolean,byte[]> result = localServerStub.lock(args[1], clientId, checksum);

if (result.containsKey(Boolean.TRUE)){

// Le fichier à été barré sur le serveur

if(result.get(Boolean.TRUE) == null){

// Si aucun fichier n'a été retourné, le fichier local est déjà à jour

System.out.println("File " + args[1] + " locked on server.");

} else {

//On met à jour le fichier local

File oldFile = new File(args[1]);

FileOutputStream fStream = new FileOutputStream(oldFile, false);

fStream.write(result.get(Boolean.TRUE));

fStream.close();

System.out.println("File " + args[1] + " updated locally and locked on server.");

}

} else {

System.out.println("File " + args[1] + " not locked on server. Use \"list\" to check if file exists or is already locked by another user.");

}

} else {

System.out.println("Please specify filename to lock on server. \n"

+ "Syntax: lock filename\n\n");

}

break;

case "push":

if(args.length >= 2){

File fileToUpdate = new File(args[1]);

byte[] fileBytes = null;

if(fileToUpdate.exists() && !fileToUpdate.isDirectory()) {

//Le fichier existe déjà localement

fileBytes = Files.readAllBytes(Paths.get(args[1]));

}

if(localServerStub.push(args[1], fileBytes, clientId)){

System.out.println("File " + args[1] + " updated and unlocked on server.");

} else {

System.out.println("File " + args[1] + " not updated on server. Use \"list\" to check if file exists and you are its owner.");

}

}

break;

default:

//Commande non reconnue

System.out.println("Command not recognized. Use \"help\" to list available commands.");

break;

}

} catch (IOException e){

e.printStackTrace();

}

}

private ServerInterface loadServerStub(String hostname) {

ServerInterface stub = null;

try {

Registry registry = LocateRegistry.getRegistry(hostname);

stub = (ServerInterface) registry.lookup("server");

} catch (NotBoundException e) {

System.out.println("Erreur: Le nom '" + e.getMessage()

+ "' n'est pas défini dans le registre.");

} catch (AccessException e) {

System.out.println("Erreur: " + e.getMessage());

} catch (RemoteException e) {

System.out.println("Erreur: " + e.getMessage());

}

return stub;

}

private void createClientID(){

try {

//On écrit le clientId généré par le serveur dans le fichier local clientInfo.xqt

PrintWriter wr = new PrintWriter("clientInfo.xqt");

clientId = localServerStub.generateclientid();

wr.println(clientId.intValue());

wr.close();

} catch (FileNotFoundException | RemoteException e) {

e.printStackTrace();

}

}

// Calcul du checksum MD5 des données passées en paramètre

private byte[] getChecksum(byte[] fileData){

MessageDigest md;

try {

md = MessageDigest.getInstance("MD5");

md.update(fileData);

return md.digest();

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

return null;

}

}

}