Paulina Czyż, Julian Nowak

# E-goat

Sieć wymiany plików peer-to-peer
https://github.com/julnow/e-goat/

**Założenia**

Projekt składa się z 2 programów: klienta i serwera. Na serwerze przechowywana jest lista plików udostępnianych przez poszczególnych klientów, wraz z informacją o klientach, którzy je posiadają. Klienci wymieniają się plikami bezpośrednio między sobą.

**Wymagania**
- Java 15

**Opis działania**

Do komunikacji pomiędzy serwerem a klientami wykorzystany jest protokół UDP (podczas komunikacji sprawdzane są sumy kontrolne poszczególnych plików, nie potrzebna jest zatem kontrola na poziomie protokołu).

1) Uruchamiamy serwer:
*$ java UDPServer*

2) Uruchamiamy klienta:
$ java UDPClient

3) Podajemy ścieżkę do katalogu, w którym znajdują się udostępniane pliki. Klient oblicza sumy kontrolne *SHA512* plików znajdujących się w katalogu i wysyła do serwera wiadomość *List of files*. Następnie wysyła do serwera listę z sumami kontrolnymi plików

3a) Do przechowywania informacji o plikach (ich sumy kontrolnej i adresu) stworzona została dodatkowa klasa *Files.*

4) Serwer po odebraniu listy wysyła odpowiedź *files added to the list*.

5) Następnie klient prosi o podanie sumy SHA512 pliku, który chcemy pobrać i wysyła ją do serwera.

6) Serwer w odpowiedzi odsyła listę wszystkich klientów (a dokładniej ich adresów), którzy zgłosili, że taki plik posiadają. W przypadku gdy nikt nie udostępnia takiego pliku serwer wysyła odpowiednią wiadomość.

7) (niezaimplementowane) Jeśli plik istnieje, klient pobiera go bezpośrednio od osoby, która go udostępnia i sprawdza czy otrzymał plik o prawidłowej sumie kontrolnej.

**Opis kodu:**

- Klasa Files

```
import java.net.InetAddress;

public class Files {

	String sha512;
	String address;
	InetAddress ip;
	int port;

	Files(String sha512, String address){
		this.sha512 = sha512;
		this.address = address;
	}

	Files(String sha512, String address, int port, InetAddress
ip){
		this.sha512 = sha512;
		this.address = address;
		this.ip = ip;
		this.port = port;

	}

	String getAddress() {
		return address;
	}

	String getSha() {
		return sha512;
	}

	int getPort() {
		return port;
	}

	InetAddress getIp() {
		return ip;
	}


}
```

Obiekty klasy *Files* zawierają adres, gdzie znajduje się kod konkretnego pliku (*address*) oraz jego hash *sha512*. Ponadto może zapisać *IP* klienta posiadającego plik i jego *port*.

- Klasa UDPClient

Klasa UDPClient zawiera funkcję, zamieniającą dany plik na strumień bitów, a następnie obliczającą *sha512* pliku:

```
private static String check(String path, MessageDigest md)
throws IOException {
        try (DigestInputStream dis = new DigestInputStream(new
FileInputStream(path), md)) {
            while (dis.read() != -1);
            md = dis.getMessageDigest();
        }

        StringBuilder result = new StringBuilder();
        for (byte b : md.digest()) {
            result.append(String.format("%02x", b));
        }
        System.out.println("Your sha512: ");
        System.out.println(result);
        return result.toString();

    }
```

Następnie klient nawiązuje połączenie z serwerem i prosi użytkownika o podanie ścieżki do folderu, który ma być udostępniony. Dla każdego z plików znajdujących się w folderze tworzony jest hash:

```
//creating connection with server
          String message = "List of files";
         InetAddress serverAddress =
InetAddress.getByName("localhost");
         System.out.println("connecting to: " + serverAddress);

       //files to be shared:
      ArrayList<Files> files = new ArrayList<Files>();
      Scanner scan = new Scanner(System.in);
      System.out.println("pls give path to shared folder:");
      String path = scan.nextLine();
      File directory = new File(path);
      while(!directory.isDirectory())
      {
           System.out.println("error! directory doesn't exist,
try again: \n");
           path = scan.nextLine();
           directory = new File(path);
      }
      //for each file in directory creating hash
      for (File fileEntry : directory.listFiles()) {
             if (fileEntry.isFile()) {
                MessageDigest md =
MessageDigest.getInstance("SHA-512"); //creating md
                 String sha = check(fileEntry.getPath(), md);
//create sha for file
                 Files file = new Files(sha,
serverAddress+"/"+fileEntry.getPath()); //adding to list of
files
                 files.add(file);
             } else {
                 System.out.println("error with " +
fileEntry.getName());
             }
         }
```

Następnie lista plików wysyłana jest na serwer jako string w formacie *sha512+'\t'+adres+'\n'* i sprawdzana jest odpowiedź serwera.

```
//creating string of files to be sent
         StringBuilder listOfFiles = new StringBuilder();
         for (Files file : files) {

listOfFiles.append(file.getSha()).append("\t").append(file.get
Address()).append("\n");
         }
         //sending list to server
         stringContents =
listOfFiles.toString().getBytes("utf8");
         sentPacket = new DatagramPacket(stringContents,
stringContents.length);
         sentPacket.setAddress(serverAddress);
         sentPacket.setPort(Config.PORT);
         socket.send(sentPacket);
         System.out.println("list of files sent to server");
         //checking response from server
         DatagramPacket receivePacket = new DatagramPacket(
new byte[Config.BUFFER_SIZE], Config.BUFFER_SIZE);
         socket.setSoTimeout(1010);
         try {
             socket.receive(receivePacket);
             System.out.println("Server received data");
         } catch (SocketTimeoutException ste) {
             System.out.println("!No response from the
server");
         }
```

Następnie pyta użytkownika, czy chce pobrać jakiś plik – jeżeli tak (‘*y*’), pobiera listę dostępnych plików:

```
//list of available files
        System.out.println("do you want to download a file?
(y/n)");
        String resp = scan.nextLine();
        if(resp.equals("y")) {
        stringContents = "y".getBytes("utf8");
        sentPacket = new DatagramPacket(stringContents,
stringContents.length);
         sentPacket.setAddress(serverAddress);
         sentPacket.setPort(Config.PORT);
         socket.send(sentPacket);
         receivePacket = new DatagramPacket( new
byte[Config.BUFFER_SIZE], Config.BUFFER_SIZE);
         socket.setSoTimeout(1010);
      try {
          socket.receive(receivePacket);
          int length = receivePacket.getLength();
           message = new String(receivePacket.getData(), 0,
length, StandardCharsets.UTF_8);
           System.out.println("available files: \n" +
message);

      } catch (SocketTimeoutException ste) {
          System.out.println("!No response from the server");
      }
```

Po wysłaniu danych klient prosi użytkownika o podanie *sha512* pliku, którego dostępność chciałby sprawdzić na serwerze, aby następnie móc go pobrać:

```
//send sha512 of file to be read
        System.out.println("enter SHA512 of a file you wish to
download: ");
        scan = new Scanner(System.in);
        String sha = scan.nextLine();
         stringContents = sha.getBytes("utf8");
         sentPacket = new DatagramPacket(stringContents,
stringContents.length);
         sentPacket.setAddress(serverAddress);
         sentPacket.setPort(Config.PORT);
         socket.send(sentPacket);
         //Read response from the server (clients who have the
file)
          receivePacket = new DatagramPacket( new
byte[Config.BUFFER_SIZE], Config.BUFFER_SIZE);
          socket.setSoTimeout(1010);
          try {
           socket.receive(receivePacket);
              int length = receivePacket.getLength();
              message = new String(receivePacket.getData(), 0,
length, StandardCharsets.UTF_8);
              if (message.equals("No such file!"))
System.out.println(message);
              else System.out.println("following clients have
the file: \n" + message);
            } catch (SocketTimeoutException ste) {
                System.out.println("No response from the
server!");
            }
```

- Klasa UDPServer

Serwer otwiera gniazdo z konkretnym portem, a następnie zaczyna nasłuchiwać:

```
//Otwarcie gniazda z okreslonym portem
        DatagramSocket datagramSocket = new
DatagramSocket(Config.PORT);
        byte[] byteResponse;
        ArrayList<Files> files = new ArrayList<Files>();

        while (true){

            DatagramPacket receivedPacket
                    = new DatagramPacket( new
byte[Config.BUFFER_SIZE], Config.BUFFER_SIZE);
            datagramSocket.receive(receivedPacket);
            int length = receivedPacket.getLength();
            String message =
                    new String(receivedPacket.getData(), 0,
length, "utf8");
```

Jeżeli otrzyma komunikat *„List of files”* zaczyna zapisywać otrzymane pliki w *ArrayList<Files> files* – odczytuje string w tym samym formacie, w jakim kodowany jest przez klienta (*sha512+’\t’+adres+’\n’*):

```
// Port i host który wysłał nam zapytanie
            InetAddress address = receivedPacket.getAddress();
            int port = receivedPacket.getPort();

            if(message.equals("List of files")) {
               DatagramPacket listPacket
                = new DatagramPacket( new
byte[Config.BUFFER_SIZE], Config.BUFFER_SIZE);
               datagramSocket.receive(listPacket);
                length = listPacket.getLength();
                message = new String(listPacket.getData(), 0,
length, "utf8");
                String[] lines = message.split("\n");
                for (String line : lines) {
                    String[] splitLine = line.split("\t");
                    files.add(new Files(splitLine[0],
splitLine[1]));
                    }
                byteResponse = "files added to the
list".getBytes("utf8");
```

Jeżeli otrzyma komunikat *‘y’*, zwraca listę dostępnych plików:

```
else if(message.equals("y")){
                StringBuilder fileList = new StringBuilder();
                for(Files file : files)
                {

     fileList.append(file.getSha()).append("\n");
                }
                byteResponse =
fileList.toString().getBytes("utf8");
            }
```

W przeciwnym razie, próbuje odebrać *sha512* pliku, który ma być pobrany, a następnie szuka go w liście plików *files* i wysyła odpowiedź składającą się z adresów wszystkich plików o tym hashu:

```
else {
                ArrayList<String> clients = new
ArrayList<String>();
                for (Files file : files) {
                     if(message.equals(file.getSha())) {
                          clients.add(file.getAddress());
                     }
                }
                StringBuilder clientsList = new
StringBuilder();
                for (String addr : clients) {
                     clientsList.append(addr).append("\n");
                }
                if(clients.isEmpty()) {
                     byteResponse = "file
unavailable".getBytes("utf8");
                }else {
                     byteResponse =
clientsList.toString().getBytes("utf8");
                }
            }

                System.out.println(message);
            Thread.sleep(1000);
            DatagramPacket response
                    = new DatagramPacket(
                        byteResponse, byteResponse.length,
address, port);

            datagramSocket.send(response);
        }
```