UDP port knocking server and client
Dominik Krawiec s16941 16c
Documentation

What is it?

It is simple port knocking server and client written in Java. Server is waiting on specified ports for proper knocking order from client. If the knocking sequence was correct then is sends to the client file specified in the code using UDP protocol on new random port. It supports many clients at the same time, any size and any type of file.

Implementation details:

Client:

Main.java

`private static List<Integer> ports` – List in which we will hold all ports number to knock.

`public static void main(String[] args) throws IOException`

The Main class is simple class in which program is checking how many argument were passed and starting the client requests.

If zero or one - program thows no enough arguments exception.

If two or more - first is address(for example 192.168.1.1) of our server and every next is number of port in order in which we want to knock from 1-65535 port numbers (for example 5555 6666 5555).

After that we are adding all ports number provided as arguments to **ports** List and saving first argument as a String adress.
Then we are calling **create(ports, adress)** method from **CreateClients** class.

In other words its responsible for taking an arguments, and saving them. Then it pass all data to **create** function.

CreateClients.java

`public static void create(List<Integer> ports, String adress) throws IOException`

This method is saving to **int last** index of last element in port List. Creates object of type UdpClient(String adress) with name **client2.** Then in loop for every port it tries to:
- start **client2.run** which is taking as an arguments number of **port** and **false** or **port** and **true** when port is last in **ports** List.

When something goes wrong it catches exception and prints error “No response from server. (Timeout 5 seconds)”
Then it is calling **client2.close()** method which close the client socket.

In the other words, it is responsible for creating UdpClient and starting threads with proper port numbers.

UdpClient.java

`private DatagramSocket socket;` – Object which is our UDP socket.
`private InetAddress address` – Address to which we will be connecting.
`private byte[] buf` – Byte array which will be our buffer to send data.

```
public UdpClient(String address) throws SocketException,
UnknownHostException
```

Constructor which creates new DatagramSocket() and saves it to **socket** variable.
Sets socket timeout to 5 seconds.
Saves String **address** given as an argument to InetAddress **address.**

```
public void run(int port, boolean last) throws IOException,
InterruptedException
```

Creates the DatagramPacket **packet** with buf data address **address** and port **port**.
Sends the **packet** to server using socket.
Then if the **last** is true:
Creates new byte array **buf2** of size 2048.
Creates DatagaramPacket **packet2** with **buf2** array, **address** and **port.**
Receives to **packet2** using **socket**.
Creates String **received** and converts received data to **received**. (port number expected)
Prints `"Authentication sucessful \nServer is connecting from port :  "+ received”`
Then it tries to:
Receive next packet which is the file name. Then it converts **packet2** data to String and creates object **file** of type File with **received** variable name.
Prints `"The following file will be downloaded: " + received`
Next it receives next packet which is expected to be file size in bytes. Converts it to String and saves as **received.**
Prints `" with size equals " + received + " bytes"`
Then it converts **received** size to integer **to.**
Now it creates ByteArrayOutputStream **outputStream.**
Then if **to** is bigger than 2048 (which is our **buf2** array size)

> We are creating a loop which is working until we reach file size. In every iteration it receives packet from server and saves its data to **outputStream**.
> Then it writes to the **file** whole **outputStream.**

Else

> It receives only one packet cut them to file size and saves in the **file**.

Prints `Download successful!\n Exiting..."`
If any exception occurs it prints StackTrace.

In the other words it is responsible for sending packets to the proper server ports, and at the last port it is responsible for receiving data about file, file itself and saving it.

```
public void close()
```

Closes socket.

Server:

Main.java

`private static List<Integer> ports` – List in which we will hold all ports number to knock.

`public static void main(String[] args)`
The Main class is simple class in which program is checking how many argument were passed and starting the client requests.
If none - program thows no enough arguments exception.
If one or more - every is number of port in order in which we want to knock from 1-65535 port numbers (for example 5555 6666 5555).

After that we are adding all ports number provided as arguments to **ports** List.
Then we are calling **create(ports)** method from **CreateServer** class.

In other words its responsible for taking an arguments, and saving them. Then it pass all data to **create** function.

CreateServer.java

`private static HashMap<String, Integer> map` – HashMap of all clients, and number of connections in proper sequence.
`private static int size` – *Size of ports List*
`private static List<Integer> order` – List of ports in excpected order.
`private static List<UdpServer> servers` – List of working servers.

`public static int sizze()`
Returns number of servers/ports.

`public static int properOrder(int d)`
Return a port number at **d** element of List.

`public static synchronized void create(List<Integer> ports)`
Creates in a loop new Objects of types **UdpServer** and adds them to **servers** List.
If the port of server is repeated it just skips it.
If any exception was thrown it prints `Cannot create socket on port " + n + "\n Exiting…"`
And closes any working servers.
If everything went ok it prints `Server has started correctly on specified ports"`

`public static synchronized int addIP(String ip)`
If **map** our ip as key it adds to its value one else it creates new **ip** key and adds 1 to its value.
Returns value.
String ip consists of ip adress and port number of client !!!

`public static synchronized void remove(String ip)`
Removes from **map** key **ip** with its value.

`public static void serverClose()`
Closes all working server. Exites the program.

UdpServer.java

`private DatagramSocket socket` - holds socket object.
`private boolean running` - Saves info about that the server should still run or not.
`private byte[] buf` - Buffer for sending/receiving udp packets.

`public UdpServer(int port) throws SocketException`
Constructor which is creating new DatagramSocket on port **port** as **socket**.

`public void run()`
This method first receives packet on port specified in **socket.** Then it saves result of calling `CreateServer.addIP` method to int **num**(this is number of correct knocking from specified ip+port).
Then it checks using `CreateServer.properOrder(num-1)==socket.getLocalPort()` statatement that the port on which server received packet is the same as proper port in order. If not it prints `"Authentication failed from " + address + " (wrong ports sequence)"`
And removes specified ip adress and port from HashMap using `CreateServer.remove` method and sets num to 0.
Then if our **num** (Number of correct knocks) is equal number of total knocks from order (`num == CreateServer.sizze()`
It tries to send to the client number of random port from 1024-65535 range and creates new Object of type `UdpDownloadServer(random, address, port)`
and starts (UdpDownloadServer extends Thread)
If any exception was thrown it prints StackTrace.

UdpDownloadServer.java

`private InetAddress address` - Holds ip adress of client.
`private int clientPort` - Holds port of client.
`private String filename` - Holds name of the file to send.
`private DatagramSocket socket` - Holds socket
`private boolean running` - Holds info about thread should still work
`private byte[] buff = new byte[2048]` - Buffer for serding data

`public UdpDownloadServer(int port, InetAddress address, int clientPort)`
Simple constructor that tries to create **socket** on port **port**. If something goes wrong it throws exception and closes all open servers. Saves address to **address** and clientPort to **clientPort**

`public void run()`
It prints information that authentication was successful.
It tries to send **filename** using **socket** and **buff** and databuffer. If something go wrong it throws exception.
Then read info about file and save it in File **copy**.
Then it tries to read all bytes from **copy** to **buff** array and makes copy of **buff** in **senddata**.
Then it saves length of **buff** in byte array **len**.
Then it tries to:
send **len** (size of the file)
saves size of file in "**to**" variable
Then if size is bigger than 2048 bytes in a loop sends packet which is byte array n*2048 to (n+1)*2048. (Dividing file into smaller pieces)
else it is sending only one packet.

Prints `"Sending to: " + address + " successful"`
If there were any exception while sending it prints StackTrace
Then changes value of running to false.

Requirements:

To compile:

JDK 1.8

To run:

JRE 1.8

How to compile?

Via commandline

1. Turn on the cmd.
2. cd to the folder in which is this file. (If you do not know what is cd command and how to use it go to https://ss64.com/nt/cd.html)
3. Write in cmd:
```
cd Source/Client
```
4. Specify your path to javac.exe file (jdk installation folder)
5. Write in cmd:
```
/path/to/jdk/javac.exe *.java
```
6. Write in cmd
```
move /path/to/Knocking/Source/Client/*.class
/path/to/Knocking/Binary/Client/*.class
```
7. Do the same for 3. cd Source/Server

(FOR LINUX INSTEAD OF “path/to/javac.exe” JUST USE “javac” and “mv” instead of “move”)

Via IntelliJ

1. Go to "How to run?" > "UNIVERSAL" >> "IntelliJ"

How to run?

UNIVERSAL

Via commandline

1. Turn on the cmd.
2. cd to the folder in which is this file. (If you do not know what is cd command and how to use it go to https://ss64.com/nt/cd.html)
3. Write in cmd:
```
cd Binary/Server
java Main
```
(You have to specify port/ports argument for example "java Main 8080 2222" in order in which you want knock)
```
cd Binary/Client
java Main
```
(You have to specify adress of server and port/ports argument for example "java Main 192.168.1.1 8080 2222" in order in which you want knock)

4. Check that the file was properly copied to Binary/Client folder!!!

Via IntelliJ

1. Turn on IntelliJ
2. Click in top left corner File > Open

3. Choose the Client or Server folder
4. Then in the right top corner click "play" button

annotation!!!

Program is working with every file and every sequence of port with many users at the same time. But to change name of file you want to send you have to directly modify the code of program. (In the task there was not information about filename or filepath as argument)
To do this you have to modify
Server => UdpDownloadServer.java => String filename = “Something.sth”