Task 2: UDP port knocking
SKJ (2019)

Introduction

The task consists in writing a pair of tasks – a server and a client, implementing "authorisation" using "UDP port knocking" (`https://en.wikipedia.org/wiki/Port knocking`).

Specification

1. The application consists of two processes: a server and a client.
2. First, the server proces must be started. This proces opens a number of UDP ports, given as parameters, and starts waiting on them for client packets. When a proper sequence of packets sent from a single origin (the same IP address and port) is notified, the server opens a randomly selected port
   - **Version 1**: server opens a TCP port
   - **Version 2**: server opens a UDP port
3. Server sends the number of port to the address, from which a proper sequence of UDP packets was received.
4. Next, the server sends to the client a **name** and **length** of the **file**, and next server sends a **content of a file** to the client. Client receives a content and store to then given name of file in its local directory.
5. After the transmission, the server goes back to listening at UDP ports or in the version supporting many clients who want to authorize at the same time, immediately (before transmission) goes to listen to the authorization packages from the other customers.
6. If a port sequence is incorrect, there will be no response from the serwer. In such case the client should terminate with an error after a timeout.

Grading and requirements

1. In order to complete the task, server and client processes together with the communication protocol should be designed and implemented, which follow the described requirements.
2. Contents of messages sent between processes is not important and is left to designer's choice.
3. The server process is executed with parameters being a sequence of UDP ports, on which consecutove "knocking" packets should be awaited. This list can have any length and numbers on it may appear multiple times (of course, a port should be opened only once). We assume, that only ports with numbers over 1024 are used.
4. If the server cannot open certain port due to its occupation by another process, it should terminate with an error.
5. The client process gets a server IP address as its first parameter. Its next parameters are UDP port numbers, to which consecutive packets should be sent. It should use random TCP and UDP ports (assigned by the system) for work.

1. Fully and correctly implemented project is worth **4 points in version 1 and 6 points in version 2**.
2. Implementing each of the following functionalities is rewarded up with the specified number of points:
    a. **at most 2 points in version 1 and at most 4 points in version 2** for implementation of the functionality described above, under assumption, that only one client can try to authorise at once.
    b. **at most 4 points in version 1 and at most 6 points in version 2** for implementation of the functionality described above, under assumption, that any number of clients can try to authorise at once.
3. The program should be written in Java, in compliance with Java 8 standard (JDK 1.8).
6. Only basic UDP/TCP socket classes can be used for implementing the UDP network functionality.
4. The projects should be saved in appropriate directories of EDUX system not later than on 30.12.2019 (the deadline can be moved by the teaching assistant).
5. The archived project file should contain:
    a. File *Documentation(studentNo)Task2.pdf*, describing what has been implemented, what hasn't, what were the difficulties and what errors are still present. In particular, the file should include the description of the designed and implemented protocol (lack of ptotocol description can signifficantly reduce the number of assigned points).
    b. Source files (JDK 1.8) (together with all the libraries used by the author that are not the part of Java standard library). The application should compile and run on PJA lab's computers.
7. NOTICE: THE DOCUMENTATION FILE IS NECESSARY. PROJECTS WITHOUT DOCUMENTATION WILL NOT BE GRADED.
7. Solutions are evaluated with respect to the correctness and compliance with the specification.
8. The quality of code and following software engineering rules can in°uence the final grade.
9. UNLESS SPECIFIED OTHERWISE, ALL THE AMBIGUITIES THAT MIGHT ARISE SHOULD BE DISCUSSED WITH TEACHING ASSISTANT. INCORRECT INTERPRETATION MAY LEAD TO REDUCING THE GRADE OR REJECTING THE SOLUTION.