Assignment 1

Computer Networks - an introduction

Author: John Herrlin
Email: jh222jx@student.lnu.se
Semester: VT2016
Area: Computer Science
Coursecode: 1DV701

Contents

1 Introduction

All virtual machines are running in Docker containers, Dockers internal network is 172.17.0.0. Base host is addressed with 172.17.0.1. The container used is the official Java, and the specific version is OpenJDK / OpenJRE 8.

2 Assignment 1.1

This assignment is based of setting up a virtual environment and send icmp (ping) packages.

Two Docker containers is set up, 172.17.0.2 and 172.17.0.3. 172.17.0.2 sends 5 icmp requests to 172.17.0.3 and gets 5 icmp replays back from target.

Figure 1: ping -c 5

3 Assignment 1.2

To handle commandline arguments i used Apache Commons CLI lib. This provides some good features for handeling cli inputs. I had to write some exception handling when types didnt match. All of this is done in the abstract class Host and is then applied to all of the hosts by inheritence, see 3.2.2.

Figure 2: Send 5 UDP package in 1 sec

3.1 Exceptions

This is the exceptions that are handled in the commandline input workflow.

NumberFormatException
    When we want a number but gets something else

NullPointerException
    When flag is given but no argument

UnrecognizedOptionException
    When option is not known

MissingArgumentException
    When flag is given but no argument

IP address
    Make sure the IP address is valid

Port number
    Make sure the port number is in the range

Message Transfer Rate
    Not negative

Seconds
    Not below 0

3.2 VG-tasks

Clairification on vg-tasks.

3.2.1 Task 1

Not implemented

3.2.2 Task 2

In the implementation there is a abstract class called Host. All of the hosts (UDPServer, UDPClient, TCPServer, TCPClient) extends Host. Host gives functionallity to validate that all input parameters are correct. It also provides a run method. All hosts override the run method to implement the specific logic for the specific host.

4 Assignment 1.3

In 4 there is a screenshot of 4 clients creating TCP connections to the server.

Figure 3: TCP server with threads, multi client connection

4.1 TCP server with small bufsize

In 4.1 there is a screenshot of an TCP server with a buffer size of 5. The client is sending a message with a size of 10. The server is handling this in a good way and assembles the stream. Returning a message that is the size of 10.


```
^Croot@bfe19bf361e9:/# java -jar code/networking-1.0-SNAPSHOT.one-jar.jar -m tcpserver -b 5
TCPServer - DEBUG - initialized: se.jherrlin.tcp.TCPServer: { mode: tcpserver, port: 4950, bufsize: 5, mtr: 1, seconds: 1, ip: 0.0.0.0 }
/172.17.0.4 says aaaaaaaaaa with a length of: 10
/172.17.0.4 says aaaaaaaaaa with a length of: 10
.......
root@485d379036dd:/# java -jar code/networking-1.0-SNAPSHOT.one-jar.jar -m tcpclient -i 172.17.0.3 -b 5 -x aaaaaaaaaa
TCPClient - DEBUG - initialized: se.jherrlin.tcp.TCPClient: { mode: tcpclient, port: 4950, bufsize: 5, mtr: 1, seconds: 1, ip: 172.17.0.3 }
Server says: aaaaaaaaaa with a lenght of: 10
root@485d379036dd:/#
```

Figure 4: TCP where bufsize is smaller than message length

4.2 UDP server with small bufsize

In 4.2 there is a screenshot of an UDP server with a buffer size of 5. The client is sendning a message with a size of 10. The UDP is not handeling this good, it cuts the message and sends back half the message to the client.


```
root@bfe19bf361e9:/# java -jar code/networking-1.0-SNAPSHOT.one-jar.jar -m udpserver -b 5
UDPServer - DEBUG - initialized: se.jherrlin.udp.UDPServer: { mode: udpserver, port: 4950, bufsize: 5, mtr: 1, seconds: 1, ip: 0.0.0.0 }
UDP echo request from 172.17.0.4 using port 4950

.......
root@485d379036dd:/# java -jar code/networking-1.0-SNAPSHOT.one-jar.jar -m udpclient -i 172.17.0.3 -b 5 -x aaaaaaaaaa
UDPClient - DEBUG - initialized: se.jherrlin.udp.UDPClient: { mode: udpclient, port: 4950, bufsize: 5, mtr: 1, seconds: 1, ip: 172.17.0.3 }
Sent and received msg not equal!
root@485d379036dd:/#
```

Figure 5: UDP where bufsize is smaller than message length

4.3 Conclusion

The reason that TCP server is handeling this in a good way it that is uses a stream. It can be refered to open a file and read or write a steam. In UDP we just get a package and nothing more, if we dont take care of the package UDP wont help us doing it.

5 Assignment 1.4, Wireshark

5.1 UDP

UDP client / server in Wireshark

5.1.1 To small buffersize

In 5.1.1 we see that the length is bigger on the way to the server than on the way back to the client

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.17.0.4	172.17.0.1	UDP	58	Source port: 4950 Destination port: 4950
2	0.000196000	172.17.0.1	172.17.0.4	UDP	47	Source port: 4950 Destination port: 4950

Figure 6: UDP server with small buffer size

5.1.2 Enought buffersize

In 5.1.2 we see that the length is the same on both ways

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.17.0.4	172.17.0.1	UDP	58	Source port: 4950 Destination port: 4950
2	0.000086000	172.17.0.1	172.17.0.4	UDP	58	Source port: 4950 Destination port: 4950

Figure 7: UDP server with enought buffer size

5.2 TCP

TCP client / server in Wireshark

5.2.1 To small buffersize

In 5.2.1 we see that there is alot of packages going back forth. TCP have the Three way handshake or Syn-Ack when esatblishing and finishing a connection stream. For every package that is send, the reciever returns an Ack to the sender that tells the sender that the reciever have got the package and the data inside is correct. If we compair the number of packages in 5.2.1 and 5.2.2 we see that 5.2.2 have less packages. The reason for this is that the buffer size is big enought to handle the data in fewer packets.

Figure 8: TCP server with small buffer size

5.2.2 Enought buffersize

Figure 9: TCP server with enought buffer size

6   Instructions

Install, compile and run instructions.

6.1   Install dependencies

This is a Maven project. Install Maven3 and OpenJDK on a Debian based system.

```
apt-get install openjdk-7-jdk
apt-get install maven
```

6.2   Compile and run

6.2.1   Compile

Navigate to project root, where the file pom.xml is located. Package the project with:

```
mvn package -e -DskipTests
```

6.2.2   Execute

Execute the code:

```
java -jar target/networking-1.0-SNAPSHOT.one-jar.jar
```

6.2.3   Docker, Optional

The code needs to be compiled with Maven before running in Docker, see 6.2.1.

```
apt-get install docker
docker run -i -t --rm -v $PWD:/code java:8 /bin/bash
java -jar code/networking-1.0-SNAPSHOT.one-jar.jar
```