TECHNICAL REPORT

SUBJECT

UDP sockets

STUDENT

Topcii Daniil

TEACHER

Crafa Nicola

Conte Alberto

UDP SOCKETS

For the creation of a udp socket, the creation of a server that accepts UDP packets from clients and responds with the current time of the server has been proposed as an exercise. To carry out the exercise, two files have been created:

SERVERS and CLIENTS.

SERVER:

- Main, here the main functions are performed such as:
  - meter charging,
  - reception of UDP packets,
  - extracting the ip address of the client,
  - usage count per client
  - sending UDP packet with time

CODE:

```
public static voidmain(String[] args) {
 // Create the server socket
 DatagramSocketserverSocket =null;
 try{
serverSocket =newDatagramSocket(PORT);

 // Load usage counters from file
 Map<InetAddress,Integer> usageCount = loadUsageCount();

 while(true) {
 // Create buffer to receive UDP packet
 byte[] receiveBuffer =new byte[BUFFER_SIZE];
 DatagramPacketreceivePacket =newDatagramPacket(receiveBuffer,
receiveBuffer.length);

 // Receive UDP packet from client
serverSocket.receive(receivePacket);

 // Extract the IP address and port number of the client
 InetAddressclientAddress = receivePacket.getAddress();
 intclientPort = receivePacket.getPort();

 // Increment the usage counter for the client
 intcount = usageCount.getOrDefault(clientAddress,0) +1;
```

```
usageCount.put(clientAddress, count);

 // If the client has exceeded the free usage limit,
 // send a message to warn that the service is paid
 Stringresponse;
 if(count > FREE_SERVICE_LIMIT) {
response ="Paid Service";
}else{
 // Send the current date and time to the client
 At your placenow =newAt your place();
response = now.toString();
}

 // Create buffer to send UDP packet
 byte[] sendBuffer = response.getBytes();
 DatagramPacketsendPacket =newDatagramPacket(sendBuffer, sendBuffer.length,
clientAddress, clientPort);

 // Send the UDP packet to the client
serverSocket.send(sendPacket);

 // Save usage counters to file
saveUsageCount(usageCount);
}
}catch(IOExceptionAnd) {
e.printStackTrace();
}finally{
 // Close the server socket if it is open
 if(serverSockets !=null) {
serverSocket.close();
}
}
}
```

- loadUsageCount
  - checking the existence of the archive file, otherwise its creation
  - reading the file

CODE:

```
private static Map<InetAddress,Integer> loadUsageCount()throws IOException{
 Map<InetAddress,Integer> usageCount =new HashMap<>();
 Pathpath = Paths.get(USAGE_COUNT_FILE);
 if(Files.exists(path)) {
 for(Stringline:Files.readAllLines(path)) {
```

```
 String[] parts = line.split(",");
 InetAddressaddress = InetAddress.getByName(parts[0]);
 intcount = Integer.parseInt(parts[1]);
usageCount.put(address, count);
}
}else{
 // Create the usage counter file if it doesn't exist
Files.createFile(path);
}
 returnusageCount;
}
```

- saveUsageCount
  - writing to the file

CODE:

```
private static voidsaveUsageCount(Map<InetAddress,Integer> usageCount)throws
IOException{
 Pathpath = Paths.get(USAGE_COUNT_FILE);
 list<String> lines =new ArrayList<>();
 for(Map.Entry<InetAddress,Integer> entries:usageCount.entrySet()) {
 InetAddressaddress = entry.getKey();
 intcount = entry.getValue();
lines.add(address.getHostAddress() +","+count);
}
Files.write(path, lines, StandardOpenOption.CREATE,
StandardOpenOption.TRUNCATE_EXISTING);
}
```

CLIENTS:

- main which performs the following functions:
  - socket creation
  - sending UDP packet
  - receiving UDP packet
  - extracting the contents of the package
  - video print of the result

CODE:

```
public static voidmain(String[] args) {
 DatagramSocketclientSocket =null;
 try{
 // Create the client socket
clientSocket =newDatagramSocket();

 // Send an empty UDP packet to the server
 byte[] sendBuffer =new byte[0];
 DatagramPacketsendPacket =newDatagramPacket(sendBuffer, sendBuffer.length,
InetAddress.getLocalHost(), PORT);
clientSocket.send(sendPacket);

 // Create buffer to receive UDP packet from server
 byte[] receiveBuffer =new byte[BUFFER_SIZE];
 DatagramPacketreceivePacket =newDatagramPacket(receiveBuffer,
receiveBuffer.length);

 // Receive UDP packet from server
clientSocket.receive(receivePacket);

 // Extract the message from the UDP packet
 Stringmessage =newString(receivePacket.getData(),0,
receivePacket.getLength());

 // Print the message received from the server
System.out.println(message);
}catch(IOExceptionAnd) {
e.printStackTrace();
}finally{
 if(clientSocket !=null) {
clientSocket.close();
}
}
}
```

As required by the track, after the tenth connection the date will not be sent but only that the service has become paid.