

LAB 03: UDP

Contents

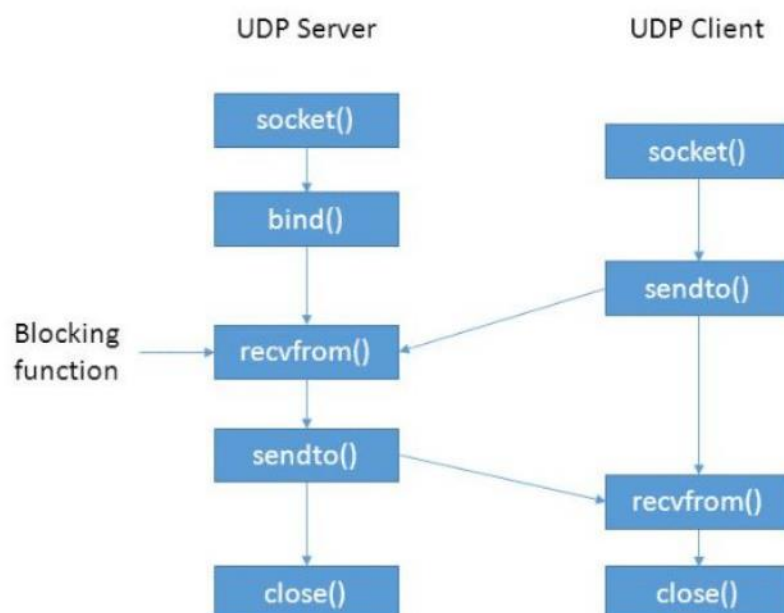
UDP Sockets	2
UDP server	2
UDP Client	7
Exercise	11

UDP Sockets

UDP server

Step	Code
1. Create a <i>DatagramSocket</i> object.	<code>DatagramSocket datagramSocket = new DatagramSocket(1234);</code>
2. Create a buffer for incoming datagrams.	<code>byte[] buffer = new byte[256];</code>
3. Create a <i>DatagramPacket</i> object for the incoming datagrams.	<code>DatagramPacket inPacket = new DatagramPacket(buffer ,buffer.length);</code>
4. Accept an incoming datagram.	<code>datagramSocket.receive(inPacket);</code>
5. Accept the sender's address and port from the packet.	<code>InetAddress clientAddress = inPacket.getAddress(); int clientPort = inPacket.getPort();</code>
6. Retrieve the data from the buffer.	<code>String message = new String(inPacket.getData(), 0,inPacket.getLength());</code>
7. Create the response datagram.	<code>DatagramPacket outPacket = new DatagramPacket(response. getBytes(),response.length(), clientAddress,clientPort);</code>
8. Send the response datagram.	<code>datagramSocket.send(outPacket);</code>
9. Close the <i>DatagramSocket</i> .	<code>datagramSocket.close();</code>

- UDP client/server diagram:



- Example

UDPEchoServer

```
public class UDPEchoServer {
    private static final int PORT = 1234;
    private static DatagramSocket datagramSocket;
    private static DatagramPacket inPacket, outPacket;
    private static byte[] buffer;

    public static void main(String[] args) {
        System.out.println("Opening Port...");
        try { // STEP 1
            datagramSocket = new DatagramSocket(PORT);
            System.out.println("Port opened successfully\n");
        } catch (IOException ioException) {
            System.out.println("Failed to open the port");
            System.exit(1);
        }
        handleClient();
    }

    private static void handleClient() {
        try {
            String msgIn, msgOut;
            int numMsgs = 0;
            InetAddress clientAddress = null;
            int clientPort;

            do {
                buffer = new byte[256]; //STEP 2
                inPacket = new DatagramPacket(buffer, buffer.length); //STEP 3
                datagramSocket.receive(inPacket); //STEP 4
                clientAddress = inPacket.getAddress(); //STEP 5
                clientPort = inPacket.getPort(); //STEP 5

                msgIn = new String(inPacket.getData(),
                                   0, inPacket.getLength()); //STEP 6
                System.out.println("Message Received");
                numMsgs++;

                msgOut = "Message " + numMsgs + ": " + msgIn;
                outPacket = new DatagramPacket(msgOut.getBytes(),
                                                msgOut.length(), clientAddress, clientPort); //STEP 7
                datagramSocket.send(outPacket); //STEP 8
            } while (true);
        } catch (IOException ioException) {
            ioException.printStackTrace();
        } finally {
            System.out.println("\n*CLOSING CONNECTION...*");
            datagramSocket.close(); //STEP 9
        }
    }
}
```

- Step-by-step code:

1. Create a class *UDPEchoServer* with *main* method and a static constant port number *PORT*.

```
public class UDPEchoServer {  
    private static final int PORT = 1234;  
  
    public static void main(String[] args) {  
  
    }  
}
```

2. Create *DatagramSocket* static variable that represents the socket for sending and receiving datagram packets.
3. Create two static variables of *DatagramPacket* that represent an incoming datagram packet and an outgoing datagram packet.
4. Create a static *byte* array that will receive incoming data.

```
public class UDPEchoServer {  
    private static final int PORT = 1234;  
    private static DatagramSocket datagramSocket;  
    private static DatagramPacket inPacket, outPacket;  
    private static byte[] buffer;  
  
    public static void main(String[] args) {
```

5. In the *main* method, setup *try* – *catch* block.
6. Inside *try* block, initialize *datagramSocket* object with port number *PORT*. Catch the exception in the *catch* block.
7. Call the function *handleClient*.

```
public static void main(String[] args) {  
    System.out.println("Opening Port...");  
    try { // STEP 1  
        datagramSocket = new DatagramSocket(PORT);  
        System.out.println("Port opened successfully\n");  
    } catch (IOException ioException) {  
        System.out.println("Failed to open the port");  
        System.exit(1);  
    }  
    handleClient();  
}
```

8. Define static method *handleClient*, and set up *try – catch – finally* block.

```
private static void handleClient() {  
    try {  
  
        } catch () {  
  
        } finally {  
  
        }  
}
```

9. Inside *try* block, define two *String* objects; one represents incoming message, and the other represents an outgoing message.
10. Inside *try* block, define an integer for counting messages.
11. Inside *try* block, define an integer for defining the client port.
12. Inside *try* block, define an *InetAddress* object for defining the IP address of the client.

```
try {  
    String msgIn, msgOut;  
    int numMsgs = 0;  
    InetAddress clientAddress = null;  
    int clientPort;
```

13. Inside *try* block, set up a *do-while* loop with the condition *true*. The loop will handle the client connections with the server.
14. Inside *do*, initialize the *byte* array to of size 256 bytes.
15. Inside *do*, initialize *inPacket* object that holds the *byte* array and its length.

```
do {  
    buffer = new byte[256]; //STEP 2  
    inPacket = new DatagramPacket(buffer,  
buffer.length); //STEP 3  
}  
while(true);
```

16. Now, the *datagramSocket* object is ready to receive an incoming packet from the client.

17. From the received packet *inPacket*, extract client's port and IP address.

```
datagramSocket.receive(inPacket); //STEP 4
clientAddress = inPacket.getAddress(); //STEP 5
clientPort = inPacket.getPort(); //STEP 5
```

18. From *inPacket*, we extract the data (client's message) into the string object *msgIn*.

19. Print "message received" to the server console, then increment the counter.

```
msgIn = new String(inPacket.getData(),
                  0, inPacket.getLength()); //STEP 6
System.out.println("Message Received");
numMsgs++;
```

20. Define the response of the server that will be sent to client into the String object *msgOut*.

21. Wrap the *msgOut* object, the message size, client's IP address, and client's port number into the datagram packet *outPacket*.

22. Send the *outPacket* to the client via *datagramSocket* object.

```
msgOut = "Message " + numMsgs + ": " + msgIn;
outPacket = new DatagramPacket(msgOut.getBytes(),
                               msgOut.length(), clientAddress,
                               clientPort); //STEP 7
datagramSocket.send(outPacket); //STEP 8
```

23. Catch the exception in *catch* block.

24. In the *finally* block, close the connection.

```
catch (IOException ioException) {
    ioException.printStackTrace();
} finally {
    System.out.println("\n*CLOSING
    CONNECTION...*");
    datagramSocket.close(); //STEP 9
}
```

UDP Client

Step		Code
1.	Create a DatagramSocket object.	<i>DatagramSocket datagramSocket = new DatagramSocket();</i>
2.	Create the outgoing datagram.	<i>DatagramPacket outPacket = new DatagramPacket(message.getBytes(), message.length(), host, PORT);</i>
3.	Send the datagram message.	<i>datagramSocket.send(outPacket);</i>
4.	Create a buffer for incoming datagrams.	<i>byte[] buffer = new byte[256];</i>
5.	Create a DatagramPacket object for the incoming datagrams.	<i>DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);</i>
6.	Accept an incoming datagram.	<i>datagramSocket.receive(inPacket);</i>
7.	Retrieve the data from the buffer.	<i>String response = new String(inPacket.getData(), 0, inPacket.getLength());</i>
8.	Close the DatagramSocket.	<i>datagramSocket.close();</i>

- Example

UDPEchoClient

```
public class UDPEchoClient {
    private static InetAddress host;
    private static final int PORT = 1234;
    private static DatagramSocket datagramSocket;
    private static DatagramPacket inPacket, outPacket;
    private static byte[] buffer;

    public static void main(String[] args) {
        try {
            host = InetAddress.getLocalHost();
        } catch (UnknownHostException ex) {
            System.out.println("Host ID not found!");
            System.exit(1);
        }
        accessServer();
    }

    private static void accessServer() {
        try {
            datagramSocket = new DatagramSocket(); //STEP 1
            Scanner userInput = new Scanner(System.in);
            String msg = "", rspns = "";
            do {
                System.out.print("Enter a message: ");
                msg = userInput.nextLine();

                if (!msg.equals("***CLOSE***")) {
                    outPacket = new DatagramPacket(msg.getBytes(),
                        0, msg.length(), host, PORT); //STEP 2
                    datagramSocket.send(outPacket); //STEP 3
                    buffer = new byte[256]; //STEP 4
                    inPacket = new DatagramPacket
                        (buffer, buffer.length); //STEP 5
                    datagramSocket.receive(inPacket); //STEP 6
                    //STEP 7
                    rspns = new String(inPacket.getData(),
                        0, inPacket.getLength());
                    System.out.println("SERVER> " + rspns);
                }
            } while (!msg.equals("***CLOSE***"));
        } catch (IOException ioException) {
            ioException.printStackTrace();
        } finally {
            System.out.println("\n*CLOSING CONNECTION...*");
            datagramSocket.close(); //STEP 8
            System.out.println();
        }
    }
}
```


- Step-by-step UDP client code:
 1. Create *UDPEchoClient* class with *main* method.
 2. Create static variables: *PORT*, *datagramSocket*, *inPacket*, *outPacket*, *buffer*, and *host*.

```
public class UDPEchoClient {
    private static InetAddress host;
    private static final int PORT = 1234;
    private static DatagramSocket datagramSocket;
    private static DatagramPacket inPacket, outPacket;
    private static byte[] buffer;

    public static void main(String[] args) {
    }
}
```

3. Inside *main*, set up *try – catch* block.
4. Inside *try* block, initialize the *host* object to get the IP address of the local machine. The local machine will act as the server and the client, so both have the same IP address.
5. Call the *accessServer* method.

```
public static void main(String[] args) {
    try {
        host = InetAddress.getLocalHost();
    } catch (UnknownHostException ex) {
        System.out.println("Host ID not found!");
        System.exit(1);
    }
    accessServer();
}
```

6. Define static method *accessServer* that will establish client's connection with the server.
7. Set up the *try – catch – finally* block.

```
private static void accessServer() {
    try {
    } catch () {

    } finally {

    }
}
```

8. Inside *try* block, initialize *DatagramSocket* object.
9. Create *Scanner* object that will get user's input from console.
10. Create two *String* objects, one for receiving message from server, and the other for sending response to the server.
11. Set up *do – while* loop, with the condition that the user's input is not *****CLOSE*****.

```
try {  
    datagramSocket = new DatagramSocket(); //STEP 1  
    Scanner userInput = new Scanner(System.in);  
    String msg = "", rspns = "";  
    do {  
  
    }  
} while (!msg.equals("***CLOSE***"));
```

12. Inside *do* block, we prompt the user to enter a message, and read it via *userInput* object.
13. After reading the user's input, we check that it is not *****CLOSE***** statement. If the user enters *****CLOSE*****, the loop terminates and the connection closes.

```
do {  
    System.out.print("Enter a message: ");  
    msg = userInput.nextLine();  
    if (!msg.equals("***CLOSE***")) {  
  
    }  
} while (!msg.equals("***CLOSE***"));
```

14. Inside *if* block, wrap the user's input (message), the message's length, the server's IP address and port into a *outPacket*.
15. Send the *outPacket* to the server via *DatagramSocket* object.

```
if (!msg.equals("***CLOSE***")) {  
    outPacket = new DatagramPacket(msg.getBytes(),  
                                    0, msg.length(), host, PORT); //STEP 2  
    datagramSocket.send(outPacket); //STEP 3
```

16. Now, the client is ready to receive the response from the server. We initialize our *buffer* array to receive the server's message.
17. Initialize the *inPacket* object to hold the *buffer* and its size.
18. Receive the *inPacket* from server via *datagramSocket* object.
19. Extract the *String* message from *inPacket*, and print the response in the console.

```
buffer = new byte[256]; //STEP 4
inPacket = new DatagramPacket(buffer,buffer.length); //STEP 5
datagramSocket.receive(inPacket); //STEP 6
//STEP 7
rspns = new String(inPacket.getData(), 0,
inPacket.getLength());
System.out.println("SERVER> " + rspns);
```

20. Catch the exception in the *catch* block.
21. In the *finally* block, terminate the connection.
22. Print the appropriate messages to the console.

```
catch (IOException ioException) {
    ioException.printStackTrace();
} finally {
    System.out.println("\n*CLOSING CONNECTION...*");
    datagramSocket.close(); //STEP 8
    System.out.println();
}
```

- Now, run the server then the client.
- ✓ CHECKPOINT: Why we did not define the closing string "****CLOSE****" in the server side in *UDPEchoServer*?

Exercise

1. Create a TCP server that accepts a username and password from a client. If the username and password are valid, the server echoes back user messages. Otherwise, the server prompts the client to enter the username and password again.
 2. Write a java application for printing the open ports of the local machine.
- ❖ Create the same server-client application explained above but with GUI.