

Name of Student : Bhavesh Vsant Laku		
Roll Number : 28		LAB Assignment Number: 2
Title of LAB Assignment: a) Implement a Server calculator containing ADD(), MUL(), SUB() etc. b) Implement a Date Time Server containing date() and time()		
DOP : 19-08-2024		DOS : 26-08-2024
CO Mapped: CO1	PO Mapped: PO3, PO5, PO7, PO12, PSO1, PSO2	Signature:

Practical No. 2

Aim: a) Implement a Server calculator containing ADD(), MUL(), SUB() etc.

b) Implement a Date Time Server containing date() and time()

Description:

Remote Procedure Call (RPC) using Datagram Sockets

1. Introduction to Remote Procedure Call (RPC)

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located on another computer in a network. RPC abstracts the communication process, making remote calls appear as local procedure calls to the programmer. This helps in building distributed systems by allowing different components of the system to interact over a network seamlessly.

2. Datagram Sockets Overview

Datagram sockets, based on the User Datagram Protocol (UDP), provide a connectionless communication mechanism. Unlike TCP sockets, UDP does not establish a connection before sending data, making it faster but less reliable. Each message is sent as a single packet, which can arrive out of order or not at all.

3. Why Use Datagram Sockets for RPC?

Using Datagram Sockets for RPC has several advantages:

- **Low Overhead:** UDP is lightweight and faster because it doesn't require connection establishment or maintenance.
- **Simple Communication:** Suitable for applications where occasional data loss is acceptable, such as real-time applications (e.g., video streaming, online gaming).
- **Scalability:** Since there's no need to maintain a connection, the server can handle many clients more easily.

However, because UDP does not guarantee delivery, additional error-checking and reliability mechanisms may need to be implemented at the application level.

4. Conceptual Model of RPC Using Datagram

In an RPC model using datagrams:

- **Client:** The client sends a request to the server by invoking a procedure, which involves sending a message (a datagram packet) to the server containing the procedure name and arguments.

- Server: The server listens for incoming datagram packets, extracts the procedure name and arguments, executes the corresponding procedure, and then sends the result back to the client in another datagram packet.

5. Designing RPC Using Datagram Sockets

A. Calculator Service Implementation Using RPC with Datagram Sockets

Step 1: Define the RPC Interface for Calculator

The Calculator Server provides the following operations:

- ADD(int a, int b) - Adds two integers.
- SUB(int a, int b) - Subtracts the second integer from the first.
- MUL(int a, int b) - Multiplies two integers.
- DIV(int a, int b) - Divides the first integer by the second.

Step 2: Implement the Calculator Server

1. Initialize the Server: Set up a DatagramSocket on a specific port (e.g., 9876).
2. Receive and Parse Client Requests: Listen for incoming requests, parse the operation and operands.
3. Perform the Arithmetic Operation: Based on the operation received, perform the calculation.
4. Send the Result Back to the Client: Send the result back using a DatagramPacket.
5. Continue Listening: The server continues to listen for new requests indefinitely.

Step 3: Implement the Calculator Client

1. Initialize the Client: Create a DatagramSocket to communicate with the server.
2. Take User Input: Continuously prompt the user for an equation (e.g., "ADD 10 5") until they choose to exit.
3. Send Request to the Server: Send the request as a DatagramPacket.
4. Receive and Display the Result: Receive the result and display it to the user.

b. DateTime Service Implementation Using RPC with Datagram Sockets

Step 1: Define the RPC Interface for DateTime

The DateTime Server provides the following operations:

- DATE() - Returns the current date in dd/MM/yyyy format.
- TIME() - Returns the current time in HH:mm:ss format.

Step 2: Implement the DateTime Server

1. Initialize the Server: Set up a DatagramSocket on a specific port (e.g., 9877).
2. Receive and Parse Client Requests: Listen for incoming requests, either "DATE" or "TIME".
3. Retrieve the Date/Time: Based on the request, retrieve the current date or time.
4. Send the Result Back to the Client: Send the result back using a DatagramPacket.
5. Continue Listening: The server continues to listen for new requests indefinitely.

Step 3: Implement the DateTime Client

1. Initialize the Client: Create a DatagramSocket to communicate with the server.
2. Take User Input: Continuously prompt the user for either "date" or "time" until they choose to exit.
3. Send Request to the Server: Send the request as a DatagramPacket.
4. Receive and Display the Result: Receive the result and display it to the user.

These steps provide a full implementation of RPC using Datagram sockets for both the Calculator and DateTime services. Each service is standalone, with its respective server and client code.

6. Example Workflow

Let's consider the workflow for an ADD(10, 5) request:

1. Client Side:

- The client prepares a packet containing the request "ADD 10 5".
- The packet is sent to the server's IP and port.
- The client waits for the server's response.

2. Server Side:

- The server receives the packet and extracts the request "ADD 10 5".
- It interprets the request, performs the addition, and prepares the response "15".
- The server sends the result back to the client.

3. Client Side:

- The client receives the packet with the result "15".
- It displays the result to the user.

7. Advantages and Disadvantages of RPC with Datagram Sockets

Advantages:

- Efficiency: Low overhead due to the lack of connection setup.

- Scalability: Suitable for a large number of clients due to connectionless nature.
- Simplicity: Easier to implement for simple, lightweight RPC operations.

Disadvantages:

- Unreliable: No guarantee of delivery, ordering, or duplicate suppression, requiring additional handling.
- No Flow Control: Unlike TCP, there's no inherent flow control, making it possible to overwhelm the server.
- Packet Size Limitation: Limited size for each request or response, potentially necessitating packet fragmentation.

8. Practical Use Cases

RPC using datagram sockets is ideal for:

- Real-Time Systems: Where speed is crucial, and occasional data loss is acceptable.
- Broadcast Systems: Where the same request is sent to multiple servers.
- Small, Stateless Services: Where the operations are simple and do not require complex state management.

A. Implement a Server calculator containing ADD(),MUL(),SUB() etc.**Code:****CalculatorServer.java**

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class CalculatorClient {

    public static void main(String[] args) throws Exception {

        DatagramSocket clientSocket = new DatagramSocket();

        InetAddress IPAddress = InetAddress.getByName("localhost");

        Scanner scanner = new Scanner(System.in);

        while (true) {

            System.out.print("Enter the equation in the format: operand1 operator operand2 (e.g., 10 + 15), or type 'exit' to quit: ");

            String equation = scanner.nextLine();

            // Check if the user wants to exit

            if (equation.equalsIgnoreCase("exit")) {

                System.out.println("Exiting the client...");

                break;

            }

            byte[] sendData = equation.getBytes();

            byte[] receiveData = new byte[1024];

            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

            clientSocket.send(sendPacket);

            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

            clientSocket.receive(receivePacket);

            String result = new String(receivePacket.getData(), 0, receivePacket.getLength());
```

```
System.out.println("Answer = " + result);  
}  
clientSocket.close();  
}  
}
```

CalculatorClient.java

```
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
  
public class CalculatorServer {  
    public static void main(String[] args) throws Exception {  
  
        DatagramSocket serverSocket = new DatagramSocket(9876);  
  
        // Notify that the server has started  
  
        System.out.println("Calculator Server is started and waiting for client  
connections...");  
  
        byte[] receiveData = new byte[1024];  
        byte[] sendData;  
  
        while (true) {  
  
            DatagramPacket receivePacket = new DatagramPacket(receiveData,  
receiveData.length);  
  
            serverSocket.receive(receivePacket);  
  
            String equation = new String(receivePacket.getData(), 0,  
receivePacket.getLength());  
  
            System.out.println("Equation Received: " + equation);  
  
            // Check if the client wants to quit  
  
            if (equation.equalsIgnoreCase("quit")) {  
  
                System.out.println("Client has exited.");  
  
                continue; // Skip processing and sending a response
```

```
}

String[] tokens = equation.split(" ");

// Validate the equation format
if (tokens.length != 3) {
    System.out.println("Invalid format.");
    sendData = "Invalid format".getBytes();
    InetAddress clientAddress = receivePacket.getAddress();
    int clientPort = receivePacket.getPort();

    DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
        clientAddress, clientPort);

    serverSocket.send(sendPacket);

    continue;
}

int operand1;
int operand2;

try {
    operand1 = Integer.parseInt(tokens[0]);
    operand2 = Integer.parseInt(tokens[2]);
} catch (NumberFormatException e) {
    System.out.println("Invalid number format.");
    sendData = "Invalid number format".getBytes();
    InetAddress clientAddress = receivePacket.getAddress();
    int clientPort = receivePacket.getPort();

    DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
        clientAddress, clientPort);

    serverSocket.send(sendPacket);

    continue;
}

String operator = tokens[1];

int result = 0;
```



```
switch (operator) {  
    case "+":  
        result = operand1 + operand2;  
        break;  
    case "-":  
        result = operand1 - operand2;  
        break;  
    case "*":  
        result = operand1 * operand2;  
        break;  
    case "/":  
        if (operand2 != 0) {  
            result = operand1 / operand2;  
        } else {  
            System.out.println("Division by zero is not allowed.");  
            sendData = "Division by zero is not allowed".getBytes();  
            InetAddress clientAddress = receivePacket.getAddress();  
            int clientPort = receivePacket.getPort();  
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,  
                clientAddress, clientPort);  
            serverSocket.send(sendPacket);  
            continue;  
        }  
        break;  
    default:  
        System.out.println("Invalid operator");  
        sendData = "Invalid operator".getBytes();  
        InetAddress clientAddress = receivePacket.getAddress();  
        int clientPort = receivePacket.getPort();
```

```

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
clientAddress, clientPort);

serverSocket.send(sendPacket);

continue;
}

sendData = String.valueOf(result).getBytes();

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
clientAddress, clientPort);

serverSocket.send(sendPacket);

System.out.println("Sending the result...");

}

}

}

```

Output:

Server

```

PS C:\Users\ADMIN\OneDrive\Desktop\New folder\DSCC> java .\CalculatorServer.java
Equation Received: 10 + 10
Sending the result...
Equation Received: 10 - 5
Sending the result...
Equation Received: 10 / 5
Sending the result...
Equation Received: 10 * 2
Sending the result...

```

Client:

```

C:\Users\ADMIN\OneDrive\Desktop\New folder\DSCC> java .\CalculatorClient.java
Enter the equation in the format: operand1 operator operand2 (e.g., 10 + 15), or type 'exit' to quit: 10 + 20
Answer = 30
Enter the equation in the format: operand1 operator operand2 (e.g., 10 + 15), or type 'exit' to quit: 20 - 10
Answer = 10
Enter the equation in the format: operand1 operator operand2 (e.g., 10 + 15), or type 'exit' to quit: 10 * 2
Answer = 20
Enter the equation in the format: operand1 operator operand2 (e.g., 10 + 15), or type 'exit' to quit: 10 / 2
Answer = 5
Enter the equation in the format: operand1 operator operand2 (e.g., 10 + 15), or type 'exit' to quit: exit
Exiting the client...

```

B. Implement a Date Time Server containing date() and time()**Code:****DateTimeServer.java**

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.text.SimpleDateFormat;
import java.util.Date;

public class DateTimeServer {

    public static void main(String[] args) throws Exception {

        // Create a DatagramSocket to listen on port 9877
        DatagramSocket serverSocket = new DatagramSocket(9877);

        System.out.println("DateTimeServer is started and listening on port 9877...");

        byte[] receiveData = new byte[1024];
        byte[] sendData;

        while (true) {

            // Create a DatagramPacket to receive data
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
                receiveData.length);

            // Receive data from the client
            serverSocket.receive(receivePacket);

            String request = new String(receivePacket.getData(), 0,
                receivePacket.getLength());

            System.out.println("Request Received: " + request);

            // Process the request
            String response;

            if (request.equalsIgnoreCase("date")) {

                response = new SimpleDateFormat("dd/MM/yyyy").format(new Date());

            } else if (request.equalsIgnoreCase("time")) {
```

```
response = new SimpleDateFormat("HH:mm:ss").format(new Date());
} else {
response = "Invalid request";
}

// Send the response to the client

sendData = response.getBytes();

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
clientAddress, clientPort);

serverSocket.send(sendPacket);

System.out.println("Sending the response...");
}
}
}
```

DateTimeClient.java

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class DateTimeClient {

public static void main(String[] args) throws Exception {

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAddress = InetAddress.getByName("localhost");

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.print("Enter 'date' or 'time' to get the current date or time, or
type 'exit' to quit: ");

String request = scanner.nextLine();
```

```
// Check if the user wants to exit
if (request.equalsIgnoreCase("exit")) {
    System.out.println("Exiting the client...");
    break;
}

byte[] sendData = request.getBytes();
byte[] receiveData = new byte[1024];

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
        IPAddress, 9877);

clientSocket.send(sendPacket);

DatagramPacket receivePacket = new DatagramPacket(receiveData,
        receiveData.length);

clientSocket.receive(receivePacket);

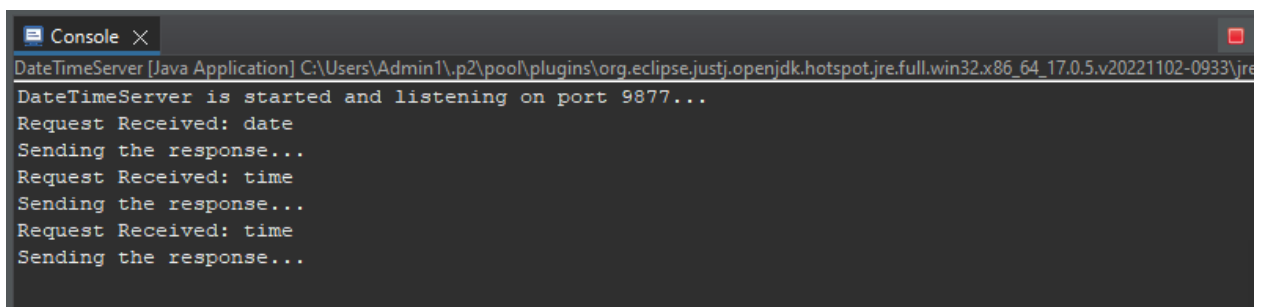
String response = new String(receivePacket.getData(), 0,
        receivePacket.getLength());

System.out.println("Result = " + response);
}

clientSocket.close();
}
}
```

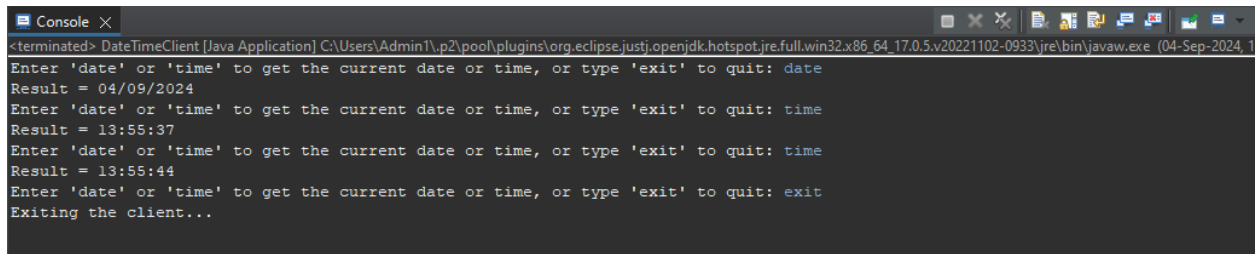
Output:

Server:



The screenshot shows a console window titled "Console" with a close button. The text inside the console is as follows:

```
DateTimeServer [Java Application] C:\Users\Admin1\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre
DateTimeServer is started and listening on port 9877...
Request Received: date
Sending the response...
Request Received: time
Sending the response...
Request Received: time
Sending the response...
```

Client:A screenshot of a Java console window titled "Console X". The window shows the execution of a Java application named "DateTimeClient". The prompt is "<terminated> DateTimeClient [Java Application] C:\Users\Admin1\p2\pooft\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (04-Sep-2024, 1". The user enters "date" and the output is "Result = 04/09/2024". The user enters "time" and the output is "Result = 13:55:37". The user enters "time" and the output is "Result = 13:55:44". The user enters "exit" and the output is "Exiting the client...".

```
<terminated> DateTimeClient [Java Application] C:\Users\Admin1\p2\pooft\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (04-Sep-2024, 1
Enter 'date' or 'time' to get the current date or time, or type 'exit' to quit: date
Result = 04/09/2024
Enter 'date' or 'time' to get the current date or time, or type 'exit' to quit: time
Result = 13:55:37
Enter 'date' or 'time' to get the current date or time, or type 'exit' to quit: time
Result = 13:55:44
Enter 'date' or 'time' to get the current date or time, or type 'exit' to quit: exit
Exiting the client...
```

Conclusion:

This implementation demonstrates how to use Datagram sockets in Java to build a simple RPC system for remote procedure calls, allowing clients to perform calculations and retrieve date/time information from a server. The approach showcases the fundamentals of network communication, emphasizing lightweight, connectionless communication via UDP, making it suitable for tasks requiring minimal overhead and real-time responses.