| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Practical Number:** 3 | |
| **Aim of Practical:**  Implement a Date Time Server containing date() and time() | | | |
| **DOP:** 8.8.23 | | **DOS:** 10.10.23 | |
| **CO Mapped:**  CO1 | **PO Mapped:**  - | **Faculty Signature:** | **Marks:** |

## 

## Practical No. 2

**Aim:** Implement a Date Time Server containing date() and time()

**Theory:**

Remote Procedure Call (RPC) using Datagram is a communication mechanism in distributed computing where programs on different computers can invoke procedures or methods on remote systems as if they were local, using the User Datagram Protocol (UDP) for communication. Here's a brief explanation:

RPC Concept: RPC is a method of interprocess communication that allows a program to cause a procedure (subroutine) to execute in another address space, typically on a remote server. The calling program can pass arguments to the remote procedure, and the procedure can return results.

Datagram: Datagram refers to the use of UDP, a connectionless transport protocol, for communication between the client and server. Unlike TCP, UDP doesn't provide reliability and ordering guarantees, making it suitable for scenarios where low latency is more critical than guaranteed delivery.

Simplicity and Efficiency: RPC over Datagram is often chosen for its simplicity and efficiency. It's well-suited for situations where a small amount of data needs to be transmitted quickly, and occasional packet loss or out-of-order delivery can be tolerated.

Stateless: UDP is stateless, meaning each packet is independent of others. This makes it a good choice for stateless services or operations that can be retried if a packet is lost.

Error Handling: RPC over Datagram typically involves less overhead for error handling and connection setup compared to RPC over TCP. However, it may require additional mechanisms for handling lost or duplicate packets if reliability is crucial.

Example Use Cases: RPC over Datagram is commonly used in scenarios such as online gaming, real-time multimedia streaming, and situations where the client and server exchange small, time-sensitive messages.

Challenges: The use of UDP introduces challenges such as packet loss, packet duplication, and dealing with out-of-order packets. To ensure reliability, additional mechanisms like acknowledgment and retransmission may be implemented at the application level.

Security: RPC over Datagram may require additional security mechanisms to protect against unauthorized access and data tampering since UDP doesn't inherently provide encryption or authentication.

**Code:**

File: DateTimeServer.java

package vesit.ajayk57.practical3;

import java.io.IOException;

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) {

try {

DatagramSocket socket = new DatagramSocket(12345);

byte[] receiveData = new byte[1024];

System.out.println("Date-Time Server is running and listening on port 12345...");

while (true) {

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

socket.receive(receivePacket);

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

// Process the request and generate a response.

String response = processRequest(request);

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

byte[] sendData = response.getBytes();

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

socket.send(sendPacket);

System.out.println("Response sent to client: " + response);

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static String processRequest(String request) {

// Check the request type and generate a response accordingly.

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

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

return "Date: " + dateFormat.format(new Date());

} else if ("time()".equals(request)) {

SimpleDateFormat timeFormat = new SimpleDateFormat("HH:mm:ss");

return "Time: " + timeFormat.format(new Date());

} else {

return "Invalid request";

}

}

}

File: DateTimeClient.java

package vesit.ajayk57.practical3;

import java.io.IOException;

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) {

try {

DatagramSocket socket = new DatagramSocket();

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

int serverPort = 12345;

// Create a scanner to read user input.

Scanner scanner = new Scanner(System.in);

while (true) {

// Prompt the user for a request (date or time).

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

String userInput = scanner.nextLine();

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

break; // Exit the loop if the user enters 'exit'.

}

// Send the user's request to the server.

byte[] sendData = userInput.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, serverPort);

socket.send(sendPacket);

// Receive the server's response.

byte[] receiveData = new byte[1024];

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

socket.receive(receivePacket);

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

System.out.println("Server Response: " + response);

}

// Close the socket and scanner.

socket.close();

scanner.close();

} catch (IOException e) {

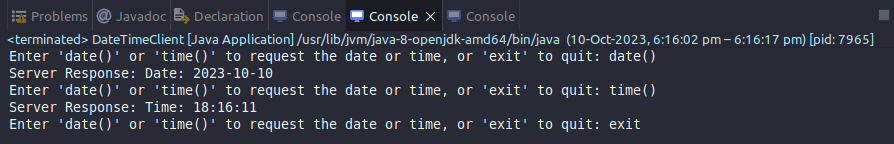
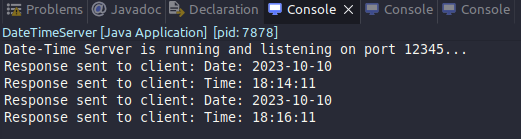
e.printStackTrace();

}

}

}

**Output:**



**Conclusion:**

I learnt how to implement a Date Time Server containing date() and time().