Assignment 1: Distributed Application using RPC – Factorial Computation

Objective

1. To understand communication between processes on different machines.
2. To design and implement a distributed application using Remote Procedure Call (RPC).

Theory: Remote Procedure Call (RPC)

Remote Procedure Call (RPC) allows a program to request a service from a program located on another computer in a network. It works similarly to a local function call, but the function executes remotely.

Key Concepts:

- **Client:** Sends request for computation.
- **Server:** Performs computation and sends back the result.
- **IDL (Interface Definition Language):** Describes the API for communication.
- **Synchronous Communication:** Client waits for the server's response.
- **Message Passing:** Used for data exchange between client and server.

Working of RPC:

1. Client prepares data and calls a remote function.
2. Data is sent over the network.
3. Server receives the request, performs the computation.
4. Result is sent back to the client.
5. Client receives and displays the result.

Code Implementation

1. Server File: factserver.py

python

CopyEdit

```
from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler


# Factorial calculation logic

class FactorialServer:

    def calculate_factorial(self, n):

        if n < 0:

            raise ValueError("Input must be a non-negative integer.")

        result = 1

        for i in range(1, n + 1):

            result *= i

        return result


# Restrict access to /RPC2 path

class RequestHandler(SimpleXMLRPCRequestHandler):

    rpc_paths = ('/RPC2',)


# Creating the XMLRPC Server

with SimpleXMLRPCServer(('localhost', 8000), requestHandler=RequestHandler) as server:

    server.register_introspection_functions()

    server.register_instance(FactorialServer())

    print("FactorialServer is ready to accept requests.")

    server.serve_forever()
```

---

2. Client File: factclient.py

python

CopyEdit

```
import xmlrpc.client


# Connect to the XMLRPC Server
with xmlrpc.client.ServerProxy("http://localhost:8000/RPC2") as proxy:
    try:
        input_value = 5  # You can change this value
        result = proxy.calculate_factorial(input_value)
        print(f"Factorial of {input_value} is: {result}")
    except Exception as e:
        print(f"Error: {e}")
```

Execution Steps

Step 1: Run the Server

- Open Command Prompt/Terminal
- Navigate to the folder containing the scripts
- Run the server:

bash

CopyEdit

```
python factserver.py
```

- Output:

vbnet

CopyEdit

```
FactorialServer is ready to accept requests.
```

Step 2: Run the Client

- Open **another** Command Prompt/Terminal

- Navigate to the same folder
- Run the client:

bash

CopyEdit

python factclient.py

- Output:

csharp

CopyEdit

Factorial of 5 is: 120

Output Explanation

The client sends 5 to the server using XML-RPC. The server calculates 5! = 120 and sends it back. The client receives and prints the result:

csharp

CopyEdit

Factorial of 5 is: 120

Conclusion

In this assignment, a distributed application was created using **RPC** where a client sends a number to a server, and the server returns the **factorial**. It demonstrated inter-process communication over a network using **Python XML-RPC**.