

ASSIGNMENT 1

PROBLEM STATEMENT: -

Design a distributed application using RPC for remote computation where client submits an integer value to the server and server calculates factorial and returns the result to the client program.

OBJECTIVE:

1. To learn and understand how to communicate between processes on different workstations.
2. To Learn and design a distributed application using Remote Procedure Call.

THEORY:

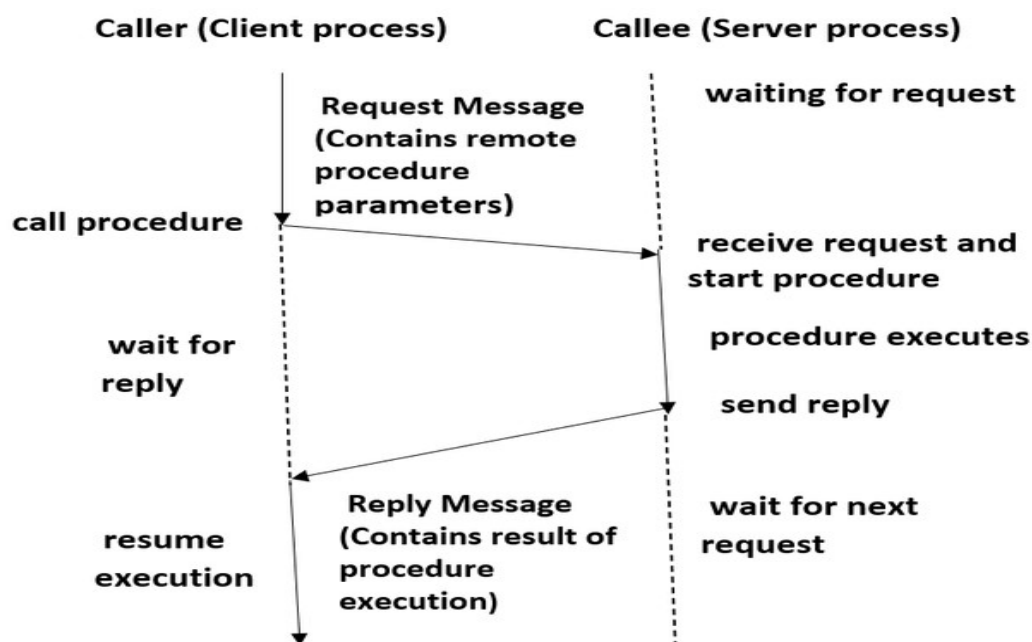
Remote Procedure Call (RPC) is a communication technology that is used by one program to make a request to another program for utilizing its service on a network without even knowing the network's details. A function call or a subroutine call are other terms for a procedure call.

It is based on the client-server concept. The client is the program that makes the request, and the server is the program that gives the service. An RPC, like a local procedure call, is based on the synchronous operation that requires the requesting application to be stopped until the remote process returns its results. Multiple RPCs can be executed concurrently by utilizing lightweight processes or threads that share the same address space. Remote Procedure Call program as often as possible utilizes the Interface Definition Language (IDL), a determination language for describing a computer program component's Application Programming Interface (API). In this circumstance, IDL acts as an interface between machines at either end of the connection, which may be running different operating systems and programming languages.

Working Procedure for RPC Model

- The process arguments are placed in a precise location by the caller when the procedure needs to be called.
- Control at that point passed to the body of the method, which is having a series of instructions.
- The procedure body is run in a recently created execution environment that has duplicates of the calling instruction's arguments.
- At the end, after the completion of the operation, the calling point gets back the control, which returns a result.

- The call to a procedure is possible only for those procedures that are not within the caller's address space because both processes (caller and callee) have distinct address space and the access is restricted to the caller's environment's data and variables from the remote procedure.
- The caller and callee processes in the RPC communicate to exchange information via the message-passing scheme.
- The first task from the server-side is to extract the procedure's parameters when a request message arrives, then the result, send a reply message, and finally wait for the next call message.
- Only one process is enabled at a certain point in time.
- The caller is not always required to be blocked.
- The asynchronous mechanism could be employed in the RPC that permits the client to work even if the server has not responded yet.
- In order to handle incoming requests, the server might create a thread that frees the server for handling consequent requests.



Implementation Steps:

1st file:Factserver.py

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler
```

```
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 to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
with SimpleXMLRPCServer(('localhost', 8000),
                        requestHandler=RequestHandler) as server:
    server.register_introspection_functions()
    # Register the FactorialServer class
    server.register_instance(FactorialServer())
    print("FactorialServer is ready to accept requests.")
    # Run the server's main loop
    server.serve_forever()
```

2nd file: Factclient.py

```
import xmlrpc.client

# Create an XML-RPC client
with xmlrpc.client.ServerProxy("http://localhost:8000/RPC2") as proxy:
    try:
        # Replace 5 with the desired integer value
        input_value = 5
        result = proxy.calculate_factorial(input_value)
        print(f"Factorial of {input_value} is: {result}")
    except Exception as e:
        print(f"Error: {e}")
```

Execution Steps

1. Open Command Prompt:

On Windows: Press Win + R, type cmd, and press Enter.

On Linux/macOS: Open a terminal.

2. Navigate to the Script's Directory:

Use the cd command to change to the directory where your Python script is located.

For example:

```
bash
cd path\to\your\script\directory
```

3. Run the Python Script:

Use the python command followed by the name of your Python script to run it.

For example:

- On Windows:

```
bash
python script.py
```

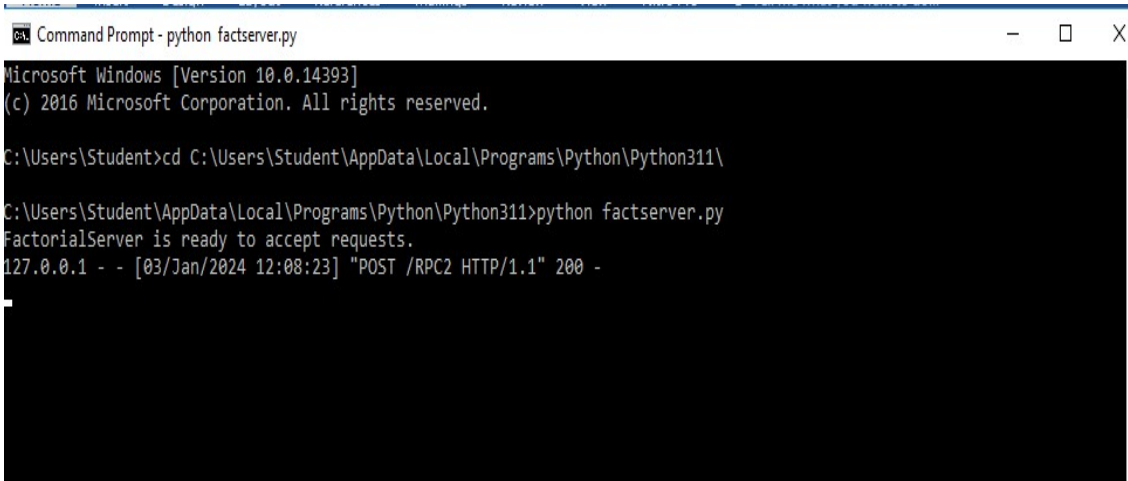
If you're using Python 3, you might need to use `python3` instead:

```
bash
python3 script.py
```

- On Linux/macOS:

```
bash
python3 script.py
```

First execute factserver.py on command prompt

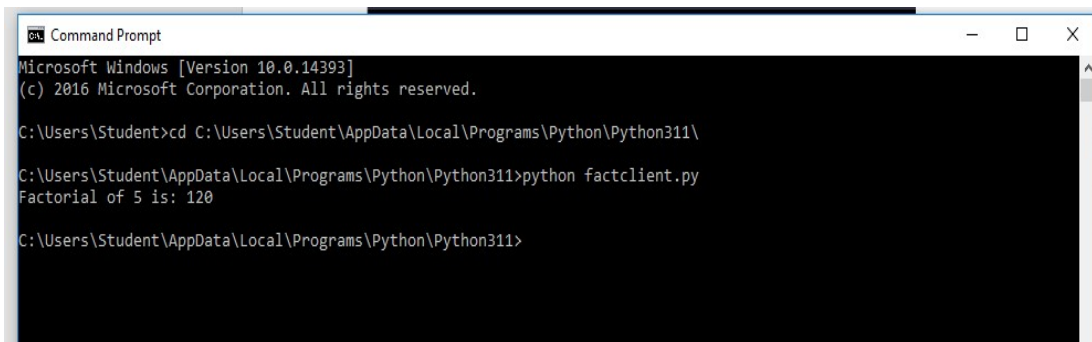


```
Command Prompt - python factserver.py
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Student>cd C:\Users\Student\AppData\Local\Programs\Python\Python311\

C:\Users\Student\AppData\Local\Programs\Python\Python311>python factserver.py
FactorialServer is ready to accept requests.
127.0.0.1 - - [03/Jan/2024 12:08:23] "POST /RPC2 HTTP/1.1" 200 -
```

Then open another Command Prompt window and execute factclient.py



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Student>cd C:\Users\Student\AppData\Local\Programs\Python\Python311\

C:\Users\Student\AppData\Local\Programs\Python\Python311>python factclient.py
Factorial of 5 is: 120

C:\Users\Student\AppData\Local\Programs\Python\Python311>
```

CONCLUSION:

In this way we have design a distributed application using RPC for remote computation where client submits an integer value to the server and server calculates factorial and returns the result to the client program.

ASSIGNMENT QUESTION

1. Define Remote Procedure Call (RPC) and explain how it enables communication between a client and a server in a distributed system?
2. Discuss the advantages and challenges of using RPC in distributed systems?
3. Describe the steps involved in implementing the RPC mechanism for the factorial computation?
4. Discuss the key components and their responsibilities in the client and server RPC implementations?
5. Explain how the client and server can exchange error messages in the RPC system?

FactServer.py

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler

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 to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
with SimpleXMLRPCServer(('localhost', 8000),
    requestHandler=RequestHandler) as server:
    server.register_introspection_functions()
    # Register the FactorialServer class
    server.register_instance(FactorialServer())
    print("FactorialServer is ready to accept requests.")
    # Run the server's main loop
    server.serve_forever()
```

FactClient.py

```
import xmlrpc.client

# Create an XML-RPC client
with xmlrpc.client.ServerProxy("http://localhost:8000/RPC2") as
proxy:
    try:
        # Replace 5 with the desired integer value
        input_value = 5
        result = proxy.calculate_factorial(input_value)
        print(f"Factorial of {input_value} is: {result}")
    except Exception as e:
        print(f"Error: {e}")
```

Output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\BE SEM VIII\CL_III_Code> python FactServer.py
FactorialServer is ready to accept requests.
127.0.0.1 - - [06/Apr/2024 10:46:46] "POST /RPC2 HTTP/1.1" 200 -
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\BE SEM VIII\CL_III_Code> python FactClient.py
Factorial of 5 is: 120
PS D:\BE SEM VIII\CL_III_Code> |
```