Experiment 02- RPC/RMI

Learning Objective: Student should be able to Built a Program for Client/server using RPC/RMI

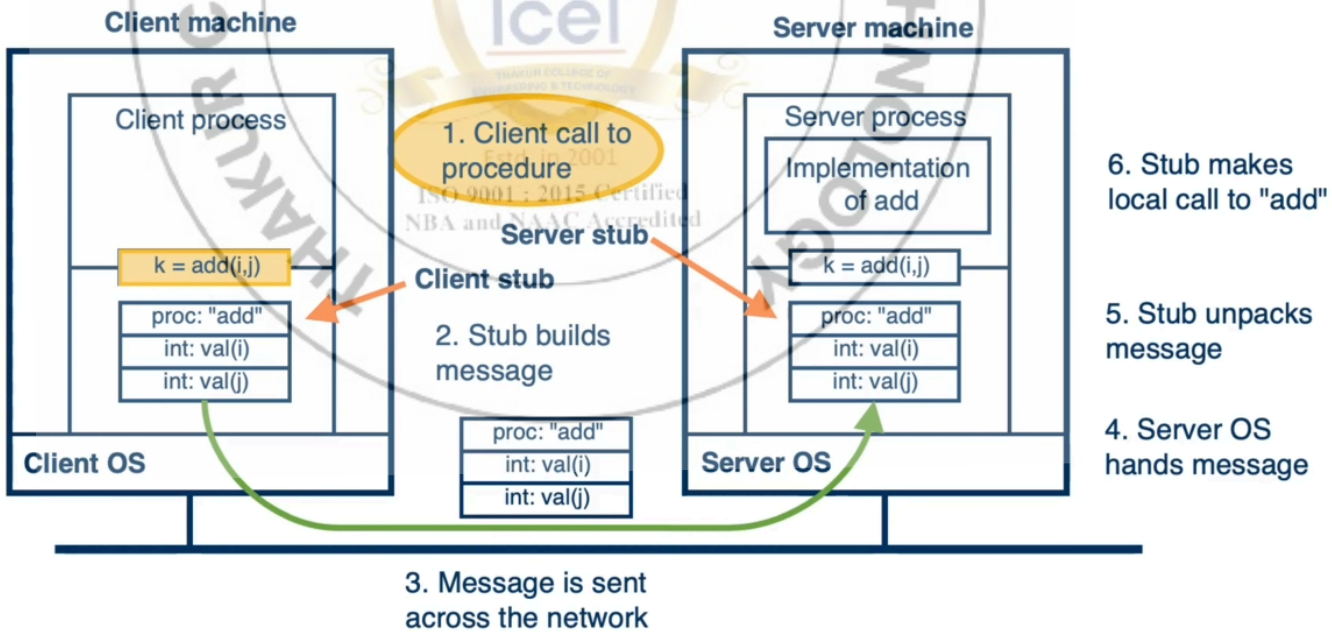
Tools: Python

Theory:

Remote Procedure call

A remote procedure call (RPC) is an inter-process communication that allows a computer program to cause a procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.

It further aims at hiding most of the intricacies of message RPC allows programs to call procedures located on other machines. But the procedures 'send' and 'receive' do not conceal the communication which leads to achieving access transparency in distributed systems.



Components of RPC

1. Client

- A Client is a user process which initiates a RPC
- The client makes a normal call that will invoke a corresponding procedure in the client stub.

2. Client Stub

Client stub is responsible for the following two tasks:

i. On receipt of a call request from the client, it packs specifications of the target procedure and arguments into a message and asks the local RPCRuntime to send it to the server stub.

ii. On receipt of the result of procedure execution, it unpacks the result and passes it to the client.

3. RPCRuntime

- Transmission of messages between Client and the server machine across the network is handled by RPCRuntime.
- It performs Retransmission, Acknowledgement, Routing and Encryption.
- RPCRuntime on Client machine receives messages containing result of procedure execution from server and sends it client stub as well as the RPCRuntime on server machine receives the same message from server stub and passes it to client machine.
- It also receives call request messages from client machine and sends it to server stub.

4. Server Stub

Server stub is similar to client stub and is responsible for the following two tasks:

i. On receipt of a call request message from the local RPCRuntime, it unpacks and makes a normal call to invoke the required procedure in the server.

ii. On receipt of the result of procedure execution from the server, it unpacks the result into a message and then asks the local RPCRuntime to send it to the client stub.

5. Server

When a call request is received from the server stub, the server executes the required procedure and returns the result to the server stub.

Working of RPC

Following steps take place during the RPC process:

Step 1- The client, the client stub, and one instance of RPC run time execute on the client machine.

Step 2- A client starts a client stub process by passing parameters in the usual way. The client stub stores within the client's own address space. It also asks the local RPC Runtime to send back to the server stub.

Step 3- In this stage, RPC accessed by the user by making regular Local Procedural Cal. RPC Runtime manages the transmission of messages between the network across client and server. It also performs the job of retransmission, acknowledgment, routing, and encryption.

Step 4- After completing the server procedure, it returns to the server stub, which packs (marshalls) the return values into a message. The server stub then sends a message back to the transport layer.

Step 5- In this step, the transport layer sends back the result message to the client transport layer, which returns back a message to the client stub.

Step 6- In this stage, the client stub demarshalls (unpack) the return parameters, in the resulting packet, and the execution process returns to the caller.

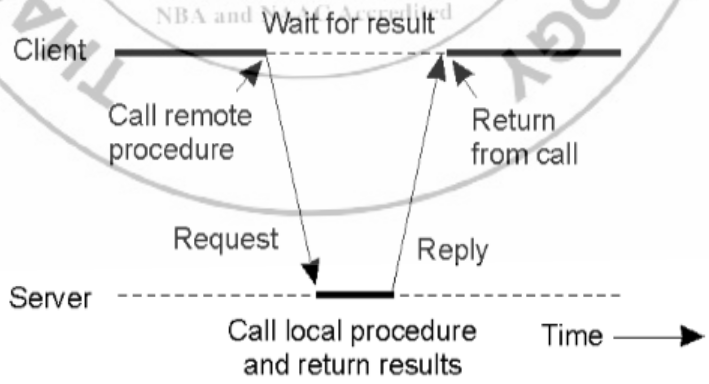


Fig. Principle of RPC between a client and server program

Code:

server.py

```
import socket
import pickle

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def handle_request(request):
    func_name = request['function']
    params = request['params']
    if func_name == 'add':
        return add(*params)
    elif func_name == 'subtract':
        return subtract(*params)
    else:
        return f"Function {func_name} not found"

if __name__ == '__main__':
    host = '0.0.0.0'
    port = 8080
    totalclients = 1
    sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    sock.bind((host, port))
    sock.listen(totalclients)
    print('Waiting for clients to connect...')
    conn, addr = sock.accept()
    print('Connected with client at', addr)
    while True:
        data = conn.recv(1024)
        if not data:
            break
        request = pickle.loads(data)  # Deserialize the
request
        print(f"Received RPC request: {request}")

        response = handle_request(request)
        conn.send(pickle.dumps(response))  #
Serialize the response
    conn.close()
    sock.close()
```

client.py

```
import socket
import pickle

def call_rpc(function, params):
    request = {'function': function, 'params':
params}
    sock.sendall(pickle.dumps(request))  # Serialize
and send the request

    response = sock.recv(1024)
    result = pickle.loads(response)  # Deserialize
the response
    return result

if __name__ == '__main__':
    host = '172.10.10.104'
    port = 8080
    sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    sock.connect((host, port))

    while True:
        function = input('Enter function name
(add/subtract): ')
        params = input('Enter parameters as comma
separated values (e.g., 3,4): ')
        params = tuple(map(int, params.split(',')))

        result = call_rpc(function, params)
        print(f'Response: {result}')

    sock.close()
```

Output:

server.py

```
PS C:\Users\Student\Desktop\Python-first> & C:/Users/Student/AppData/Local/Microsoft/
WindowsApps/python3.11.exe c:/Users/Student/Desktop/Python-first/rpc-server.py
Waiting for clients to connect...
Connected with client at ('172.10.10.104', 52591)
Received RPC request: {'function': 'add', 'params': (5, 14)}
Received RPC request: {'function': 'subtract', 'params': (27, 24)}
```

client.py

```
PS C:\Users\Student\Desktop\Python-first> python .\rpc-client.py
Enter function name (add/subtract): add
Enter parameters as comma separated values (e.g., 3,4): 5,14
Response: 19
Enter function name (add/subtract): subtract
Enter parameters as comma separated values (e.g., 3,4): 27,24
Response: 3
```

Learning Outcomes: The student should have the ability to:
LO 1: Understand the concept of Remote Procedure and Remote Method Invocation
LO 2: Apply RPC, RMI for communication.

Course Outcomes: Upon completion of the course students will be able to understand communication using Remote Method Invocation and Remote Procedure Call.

Conclusion:

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	Total
Marks Obtained				