Experiment 2

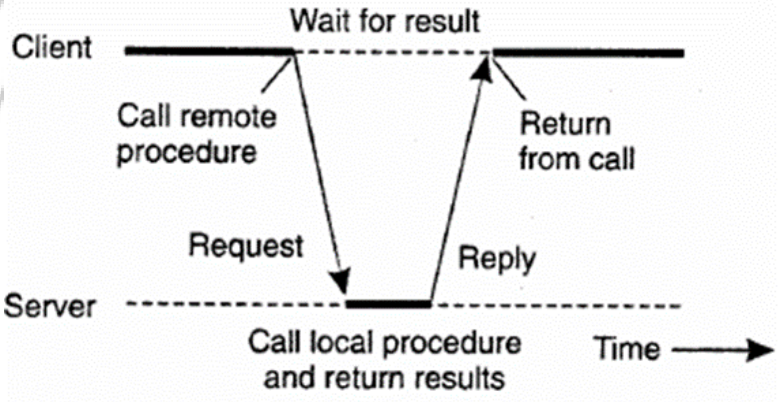
Aim: Build a program for client/server using RPC/RMI

Tools: Java / python

Theory:

Remote Procedure Call (RPC)

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.



RPC implementation steps using python:

A popular choice for RPC in Python is gRPC, which is a high-performance RPC framework. gRPC supports multiple programming languages, making it easier to achieve communication between different platforms.

1. Defining the Service Interface:

    a. Create a .proto file to define the service interface, including remote methods and message types.
    b. Use protobuf syntax to define the service and message formats.

example.proto

```
1   syntax = "proto3";
2
3   package example;
4
5   service RemoteService {
6     rpc RemoteMethod (Request) returns (Response);
7   }
8
9   message Request {
10    string message = 1;
11  }
12
13  message Response {
14    string message = 1;
15  }
```

2. Generating Code:
    a. Use the gRPC protocol compiler (protoc) to generate client and server code in the desired programming languages from the .proto file.
    b. The generated code includes stubs for the client and service implementation for the server.
    c. Command for compilation: python -m grpc_tools.protoc -I. --python_out=. –grpc_python_out=. example.proto
3. Implementing the Server:
    a. Implement the server-side logic by defining the service methods and handling client requests.
    b. Start the gRPC server, specifying the port on which it will listen for incoming requests.
4. Creating the Client:
    a. Implement the client-side logic to communicate with the server using the generated client stub.
    b. Establish a connection to the server and invoke remote methods as needed.
5. Testing the Application:
    a. Start the server and ensure it is listening on the specified port.
    b. Run the client application and observe the communication between client and server.
    c. Verify that the expected data is transmitted between client and server.

Code:

server.py

server.py > RemoteServiceServicer > RemoteMethod > request

```python
import grpc
import example_pb2
import example_pb2_grpc
from concurrent import futures

class RemoteServiceServicer(example_pb2_grpc.RemoteServiceServicer):
    def RemoteMethod(self, request, context):
        return example_pb2.Response(message=f"Received: {request.message}")

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    example_pb2_grpc.add_RemoteServiceServicer_to_server(RemoteServiceServicer(), server)
    server.add_insecure_port("[::]:50051")
    server.start()
    server.wait_for_termination()

if __name__ == "__main__":
    serve()

```

client.py

client.py > ...

```python
import grpc
import example_pb2
import example_pb2_grpc

def run():
    with grpc.insecure_channel("175.175.2.8:50051") as channel:
        stub = example_pb2_grpc.RemoteServiceStub(channel)
        response = stub.RemoteMethod(example_pb2.Request(message="Hello, Server!"))
    print(f"Client received: {response.message}")

if __name__ == "__main__":
    run()

```

Output:

Received message on client

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\comp lal-325\Desktop\implementing-procedure-call> python client.py
Client received: Received: Hello, Server!
PS C:\Users\comp lal-325\Desktop\implementing-procedure-call>
```

Learning Objective: Students should have the ability to

LO1: Understand the difference between RPC/RMI implementation in distributed systems

LO2: Understand the implementation of RPC/RMI, emphasizing abstraction and message exchange in distributed applications

Course Outcomes: Upon Completion of this students will be able to understand the concept of RPC/RMI

Conclusion: By implementing remote method invocation (RMI/RPC), learners have gained insight into the foundational concepts of distributed systems, including message passing and network communication protocols. This practical describes the significance of remote service invocation in building scalable and interconnected applications.

For Faculty Use:

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