BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to University of Mumbai)
[Knowledge is Nectar]

Department of Computer Engineering

Course - Distributed Computing (DC)

UID	2023800024
Name	Hitesh Ghanchi
Class and Batch	TE Computer Science and Engineering - Batch A2
Date	29/08/24
Lab #	3
Aim	Client-Server Communication using RPC.
Objective	1. Students must implement RPC using any complex function of their choice. Examples include: ● Addition of the first 100 prime numbers ● Multiplication of the first 10 natural numbers (factorial) ● Sum of the first 100 natural numbers ● Power function power(x, y) ● Modulus function modulus(x, y) 2. Implement the chosen function in all three RPC frameworks: ● XML-RPC ● gRPC ● Pyro5 3. Perform three complete practical implementations of RPC: ● One using XML-RPC ● One using gRPC ● One using Pyro5
Theory	RPC simplifies distributed programming by allowing a program to execute a function on a remote server as if it were a local function, abstracting away the underlying network complexities. This is a significant improvement over traditional socket-based or API-based communication, which requires a programmer to manually handle message serialization, network transport, and deserialization. The process involves a **client stub** that marshals (packs) the function parameters into a message and an **RPC runtime system** that handles the network transport. On the server side, a **server stub** unmarshals (unpacks) the message and invokes the actual procedure. The result is then sent back to the client using the same process in reverse. This paradigm shifts the focus from explicit message-oriented communication to a more intuitive, function-oriented approach.

This lab explores three distinct RPC frameworks to highlight their different design philosophies and use cases:

- **XML-RPC:** An older, text-based framework that is easy to understand and debug but is inefficient due to the verbose nature of XML.
- **Pyro5:** A Python-specific framework that prioritizes ease of use and rapid development for homogeneous (Python-to-Python) environments.
- **gRPC:** A modern, high-performance, and language-agnostic framework that uses a compact, binary serialization format (Protocol Buffers) and is well-suited for building scalable, cross-platform microservices.

Implementation / Code

XML

1) **server.py**

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

class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

server =  SimpleXMLRPCServer(('localhost', 8000),
                        requestHandler=RequestHandler,
                        allow_none=True)

def factorial(n):
        """Calculate factorial of n"""
        print(f"Request: factorial({n})")
        try:
            result = math.factorial(n)
            print(f"Response: {result}")
            return result
        except ValueError as e:
            return f"Error: {e}"

server.register_function(factorial, 'factorial')

def power(x, y):
        """Calculate x raised to power y"""
        print(f"Request: power({x}, {y})")
        result = math.pow(x, y)
        print(f"Response: {result}")
        return result

server.register_function(power, 'power')
```

```
def sum_first_n_naturals(n):
        """Calculate sum of first n natural numbers"""
        print(f"Request: sum_first_n_naturals({n})")
        if n < 0:
            return "Error: n must be non-negative"
        result = n * (n + 1) // 2
        print(f"Response: {result}")
        return result

server.register_function(sum_first_n_naturals, 'sum_first_n_naturals')

print("XML-RPC Server is running on http://localhost:8000")
print("Available functions:")
print("- factorial(n)")
print("- power(x, y)")
print("- sum_first_n_naturals(n)")
print("Press Ctrl+C to stop the server...")

server.serve_forever()
```

2) **client.py**

```
import xmlrpc.client
import time

def measure_performance(proxy, function_name, *args):
    try:
        start_time = time.perf_counter()

        result = getattr(proxy, function_name)(*args)

        end_time = time.perf_counter()
        elapsed_time = (end_time - start_time) * 1000
        print(f"{function_name}({', '.join(map(str, args))}) = {result}")
        print(f"Time taken: {elapsed_time:.3f} ms")
        print("-" * 40)

        return elapsed_time

    except Exception as e:
        print(f"Error calling {function_name}: {e}")
        return None

def main():
    proxy = xmlrpc.client.ServerProxy("http://localhost:8000")

    times = []
    print("Testing XML-RPC functions...")
    print("=" * 40)

    time_factorial = measure_performance(proxy, 'factorial', 10)
    if time_factorial:
        times.append(time_factorial)

    time_power = measure_performance(proxy, 'power', 2, 8)
    if time_power:
        times.append(time_power)

    time_sum = measure_performance(proxy, 'sum_first_n_naturals', 100)
    if time_sum:
        times.append(time_sum)

    if times:
        avg_time = sum(times) / len(times)
        print(f"Average execution time: {avg_time:.3f} ms")

    print("\nTesting multiple requests (100 calls)...")
    multiple_requests_time = measure_multiple_requests(proxy,
'sum_first_n_naturals', 50, 100)
    print(f"Total time for 100 requests: {multiple_requests_time:.3f} ms")
    print(f"Average per request: {multiple_requests_time/100:.3f} ms")

def measure_multiple_requests(proxy, function_name, arg, num_requests):
    """Measure time for multiple consecutive requests"""
    start_time = time.perf_counter()

    for i in range(num_requests):
        try:
            getattr(proxy, function_name)(arg)
        except Exception as e:
            print(f"Error in request {i+1}: {e}")
            break

    end_time = time.perf_counter()
    return (end_time - start_time) * 1000

if __name__ == "__main__":
    main()
```

PART pyro

1) **client.py**

```
import Pyro5.api
import time

def measure_performance(proxy, method_name, *args, num_requests=1):
    """Measure execution time of a remote function call"""
    try:
        start_time = time.perf_counter()

        # Call the remote function multiple times for averaging
        for _ in range(num_requests):
            result = getattr(proxy, method_name)(*args)

        end_time = time.perf_counter()
        elapsed_time = (end_time - start_time) * 1000  # Convert to milliseconds

        if num_requests > 1:
            avg_time = elapsed_time / num_requests
            print(f"{method_name} - {num_requests} requests: {elapsed_time:.3f} ms
total, {avg_time:.3f} ms avg")
            return avg_time
        else:
            print(f"{method_name}{args} = {result}")
            print(f"Time taken: {elapsed_time:.3f} ms")
            print("-" * 40)
            return elapsed_time

    except Exception as e:
        print(f"Error calling {method_name}: {e}")
        return None

def main():
    try:
        # Locate the Pyro5 name server (it's running on localhost:9090)
        with Pyro5.api.locate_ns() as ns:
            print("Found name server!")

            # Look up our calculator service
            uri = ns.lookup("calculator")
            print(f"Found calculator service: {uri}")

            # Create a proxy to the remote object
            proxy = Pyro5.api.Proxy(uri)

            print("Testing Pyro5 functions...")
            print("=" * 50)

            # Test individual functions
            times = []

            # Test factorial
            time_factorial = measure_performance(proxy, 'factorial', 10)
            if time_factorial:
                times.append(time_factorial)

            # Test power
            time_power = measure_performance(proxy, 'power', 2, 8)
            if time_power:
                times.append(time_power)

            # Test sum of naturals
            time_sum = measure_performance(proxy, 'sum_first_n_naturals', 100)
            if time_sum:
                times.append(time_sum)

            # Calculate average time
            if times:
                avg_time = sum(times) / len(times)
                print(f"Average execution time: {avg_time:.3f} ms")

            # Test multiple requests for performance analysis
            print("\n" + "=" * 50)
            print("Performance test with multiple requests:")

            # Test 100 requests for sum function
            measure_performance(proxy, 'sum_first_n_naturals', 50,
num_requests=100)

    except Pyro5.errors.NamingError:
        print("Error: Could not find the name server. Make sure it's running with 'python
-m Pyro5.nameserver'")
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

2) **server.py**

```
import Pyro5.api
import math
import time
```

```
@Pyro5.api.expose
class Calculator(object):
    def factorial(self, n):
        """Calculate factorial of n"""
        print(f"Request: factorial({n})")
        try:
            result = math.factorial(n)
            print(f"Response: {result}")
            return result
        except ValueError as e:
            return f"Error: {e}"

    def power(self, x, y):
        """Calculate x raised to power y"""
        print(f"Request: power({x}, {y})")
        result = math.pow(x, y)
        print(f"Response: {result}")
        return result

    def sum_first_n_naturals(self, n):
        """Calculate sum of first n natural numbers"""
        print(f"Request: sum_first_n_naturals({n})")
        if n < 0:
            return "Error: n must be non-negative"
        result = n * (n + 1) // 2
        print(f"Response: {result}")
        return result

def main():
    # Create a Pyro5 daemon (server)
    daemon = Pyro5.api.Daemon()

    # Register our Calculator class with the daemon
    uri = daemon.register(Calculator)

    # Register with the name server
    with Pyro5.api.locate_ns() as ns:
        ns.register("calculator", uri)

    print("Pyro5 Server is running...")
    print(f"Object URI: {uri}")
    print("Available functions:")
    print("- factorial(n)")
    print("- power(x, y)")
    print("- sum_first_n_naturals(n)")
    print("Press Ctrl+C to stop the server...")

    # Start the server event loop
    try:
        daemon.requestLoop()
    except KeyboardInterrupt:
        print("Shutting down server...")
        daemon.close()

if __name__ == "__main__":
    main()
```

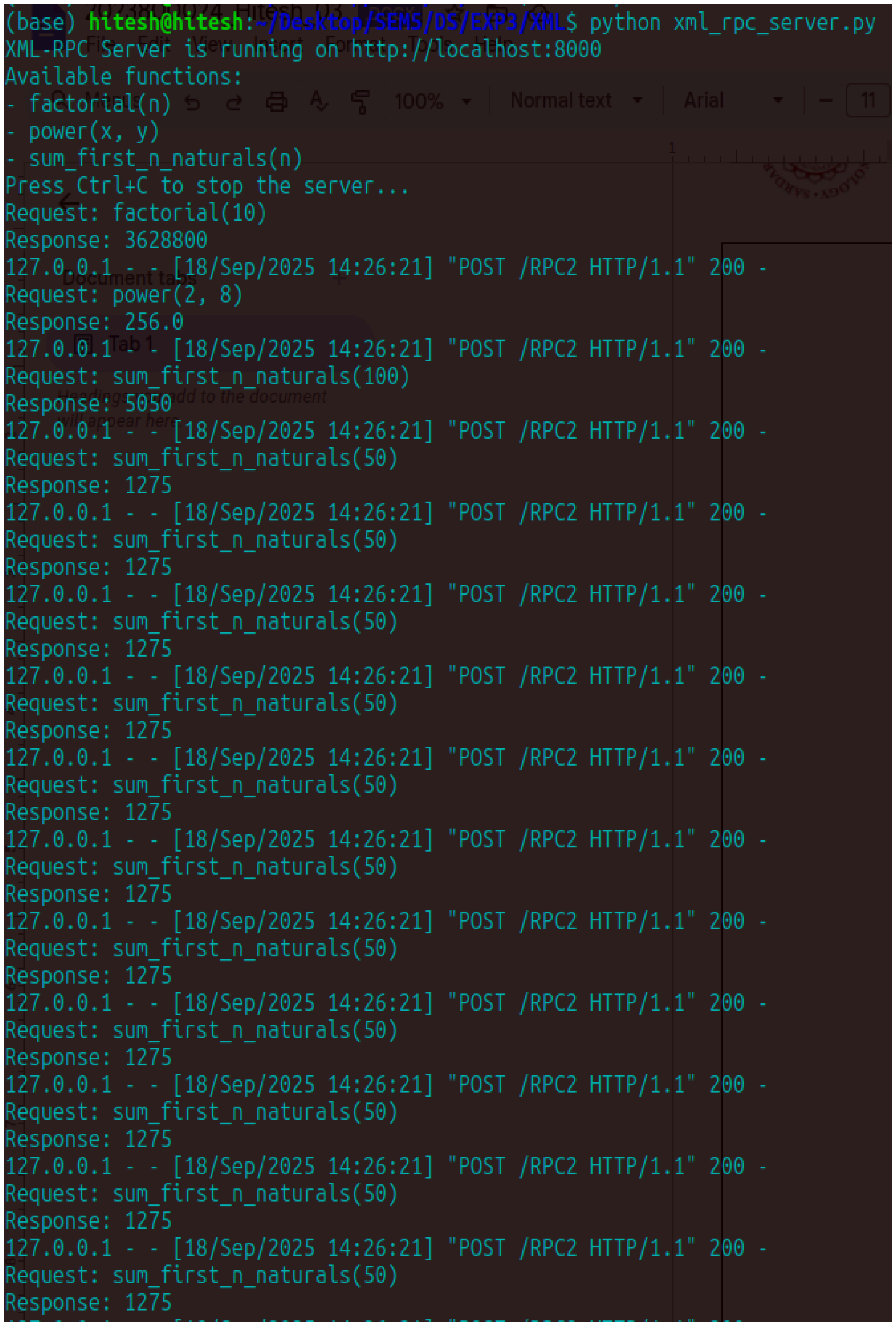
PART grpc

1) **server.py**

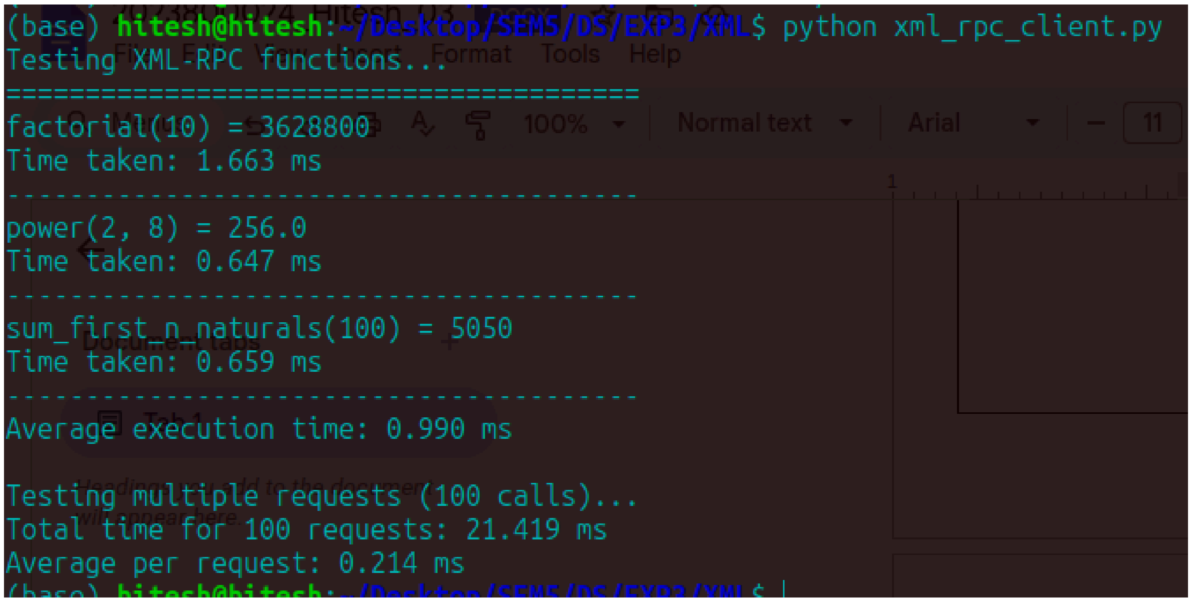
```
import grpc
from concurrent import futures
import time
import math
import calculator_pb2
import calculator_pb2_grpc

class CalculatorServicer(calculator_pb2_grpc.CalculatorServicer):
    def Factorial(self, request, context):
        """Calculate factorial of a number"""
        print(f"Request: factorial({request.number})")
        try:
            result = math.factorial(request.number)
            print(f"Response: {result}")
            return calculator_pb2.NumberResponse(result=result)
        except ValueError as e:
            return calculator_pb2.NumberResponse(error=str(e))

    def Power(self, request, context):
        """Calculate power of base to exponent"""
        print(f"Request: power({request.base}, {request.exponent})")
        result = pow(request.base, request.exponent)
        print(f"Response: {result}")
        return calculator_pb2.NumberResponse(result=result)

    def SumFirstNaturals(self, request, context):
        """Calculate sum of first n natural numbers"""
        print(f"Request: sum_first_naturals({request.number})")
        if request.number < 0:
            return calculator_pb2.NumberResponse(error="n must be non-negative")
        result = request.number * (request.number + 1) // 2
        print(f"Response: {result}")
        return calculator_pb2.NumberResponse(result=result)

def serve():
    # Create a gRPC server with thread pool
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))

    # Add our service to the server
    calculator_pb2_grpc.add_CalculatorServicer_to_server(CalculatorServicer(),
server)

    # Listen on port 50051
    server.add_insecure_port('[::]:50051')
    server.start()
    print("gRPC Server running on port 50051...")
    print("Available functions:")
    print("- Factorial(n)")
    print("- Power(base, exponent)")
    print("- SumFirstNaturals(n)")

    # Keep server running
    try:
        while True:
            time.sleep(86400)  # One day in seconds
    except KeyboardInterrupt:
        server.stop(0)

if __name__ == '__main__':
    serve()
```

2) **client.py**

```
 import grpc
import time
import calculator_pb2
import calculator_pb2_grpc

def measure_performance(stub, method_name, request, num_requests=1):
    """Measure execution time of a remote function call"""
    try:
        start_time = time.perf_counter()

        # Call the remote function multiple times for averaging
        for _ in range(num_requests):
            if method_name == 'Factorial':
                response = stub.Factorial(request)
            elif method_name == 'Power':
                response = stub.Power(request)
            elif method_name == 'SumFirstNaturals':
                response = stub.SumFirstNaturals(request)

        end_time = time.perf_counter()
        elapsed_time = (end_time - start_time) * 1000  # Convert to milliseconds

        if num_requests > 1:
            avg_time = elapsed_time / num_requests
            print(f"{method_name} - {num_requests} requests: {elapsed_time:.3f} ms
total, {avg_time:.3f} ms avg")
            return avg_time
        else:
            print(f"{method_name} - Result: {response.result}, Time: {elapsed_time:.3f}
ms")
            if response.error:
                print(f"Error: {response.error}")
            return elapsed_time

    except grpc.RpcError as e:
        print(f"gRPC Error calling {method_name}: {e}")
        return None
    except Exception as e:
        print(f"Error calling {method_name}: {e}")
        return None

def main():
    # Create a channel and stub
    channel = grpc.insecure_channel('localhost:50051')
    stub = calculator_pb2_grpc.CalculatorStub(channel)

    print("Testing gRPC functions...")
    print("=" * 50)

    # Test individual functions
    times = []

    # Test factorial
    factorial_request = calculator_pb2.NumberRequest(number=10)
    time_factorial = measure_performance(stub, 'Factorial', factorial_request)
    if time_factorial:
        times.append(time_factorial)

    # Test power
    power_request = calculator_pb2.PowerRequest(base=2, exponent=8)
    time_power = measure_performance(stub, 'Power', power_request)
    if time_power:
        times.append(time_power)

    # Test sum of naturals
    sum_request = calculator_pb2.NumberRequest(number=100)
    time_sum = measure_performance(stub, 'SumFirstNaturals', sum_request)
    if time_sum:
        times.append(time_sum)

    # Calculate average time
    if times:
        avg_time = sum(times) / len(times)
        print(f"\nAverage execution time: {avg_time:.3f} ms")

    # Test multiple requests for performance analysis
    print("\n" + "=" * 50)
    print("Performance test with multiple requests:")

    # Test 100 requests for sum function
    measure_performance(stub, 'SumFirstNaturals', sum_request, 100)

    # Close the channel
    channel.close()

if __name__ == "__main__":
    main()
```

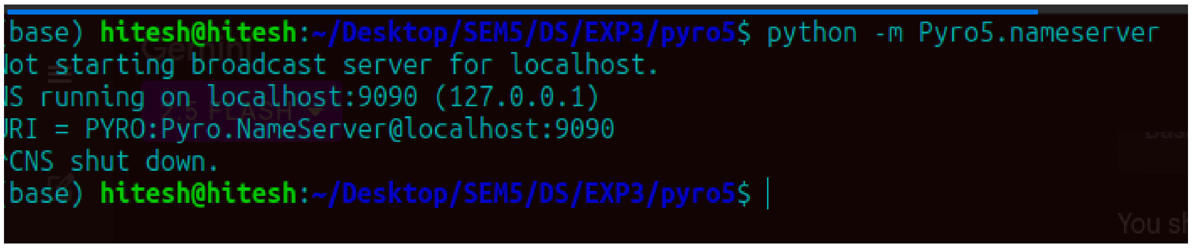
Output	PART XML 

```
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/XML$ python xml_rpc_server.py
XML-RPC Server is running on http://localhost:8000
Available functions:
- factorial(n)
- power(x, y)
- sum_first_n_naturals(n)
Press Ctrl+C to stop the server...
Request: factorial(10)
Response: 3628800
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: power(2, 8)
Response: 256.0
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(100)
Response: 5050
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
127.0.0.1 - - [18/Sep/2025 14:26:21] "POST /RPC2 HTTP/1.1" 200 -
Request: sum_first_n_naturals(50)
Response: 1275
```

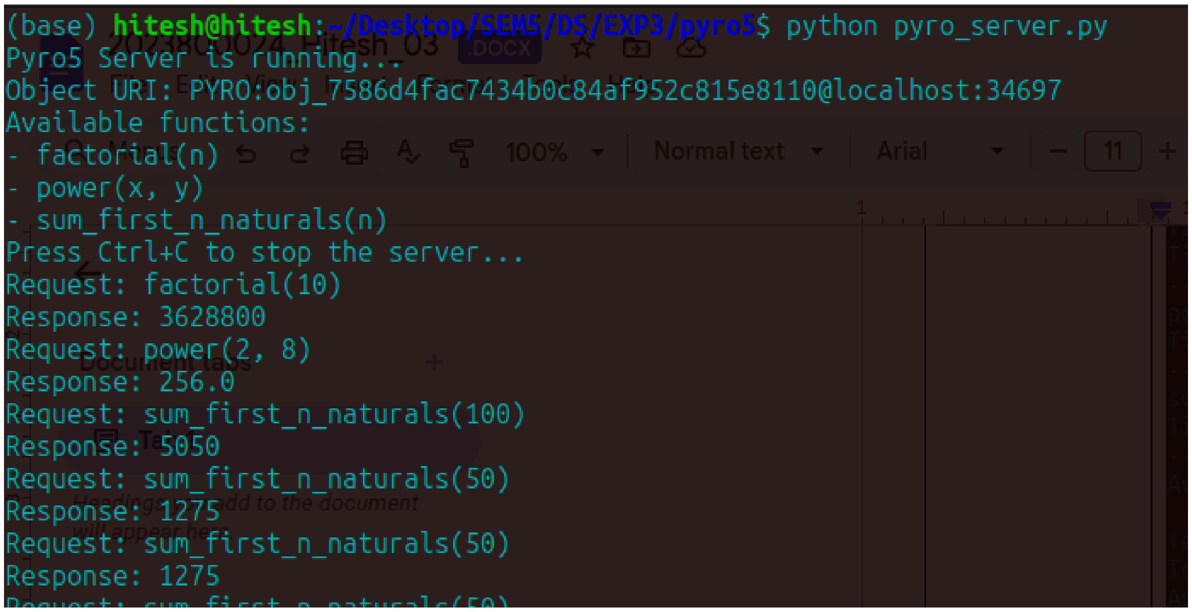


```
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/XML$ python xml_rpc_client.py
Testing XML-RPC functions...
========================================
factorial(10) = 3628800
Time taken: 1.663 ms
----------------------------------------
power(2, 8) = 256.0
Time taken: 0.647 ms
----------------------------------------
sum_first_n_naturals(100) = 5050
Time taken: 0.659 ms
----------------------------------------
Average execution time: 0.990 ms

Testing multiple requests (100 calls)...
Total time for 100 requests: 21.419 ms
Average per request: 0.214 ms
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/XML$ |
```

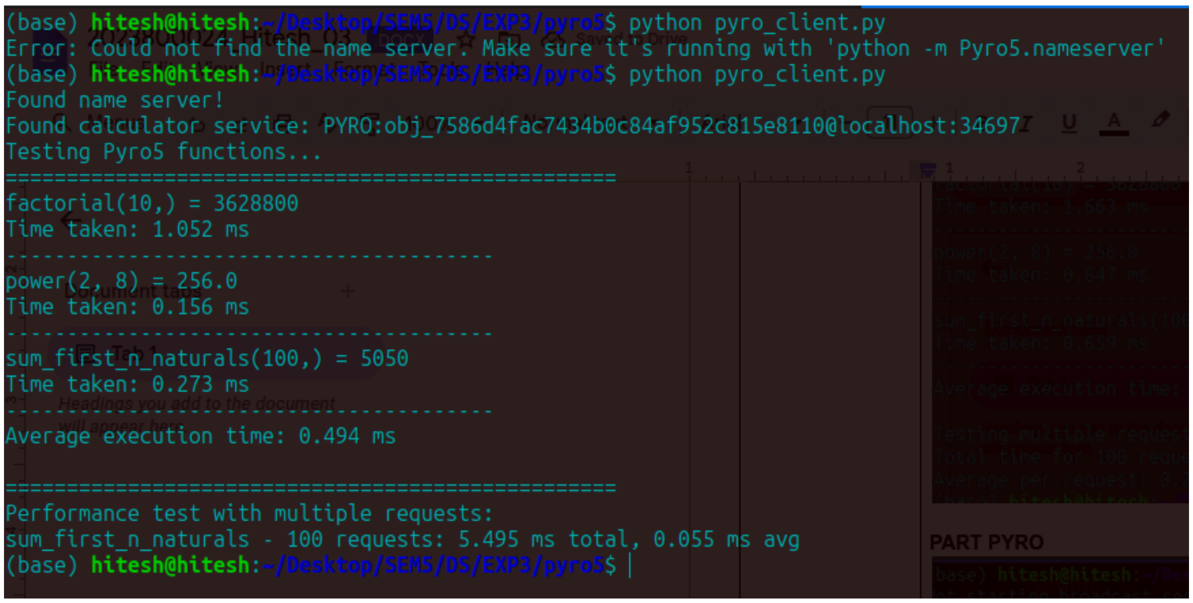
PART PYRO



```
base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/pyro5$ python -m Pyro5.nameserver
Not starting broadcast server for localhost.
NS running on localhost:9090 (127.0.0.1)
URI = PYRO:Pyro.NameServer@localhost:9090
^CNS shut down.
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/pyro5$ |
```

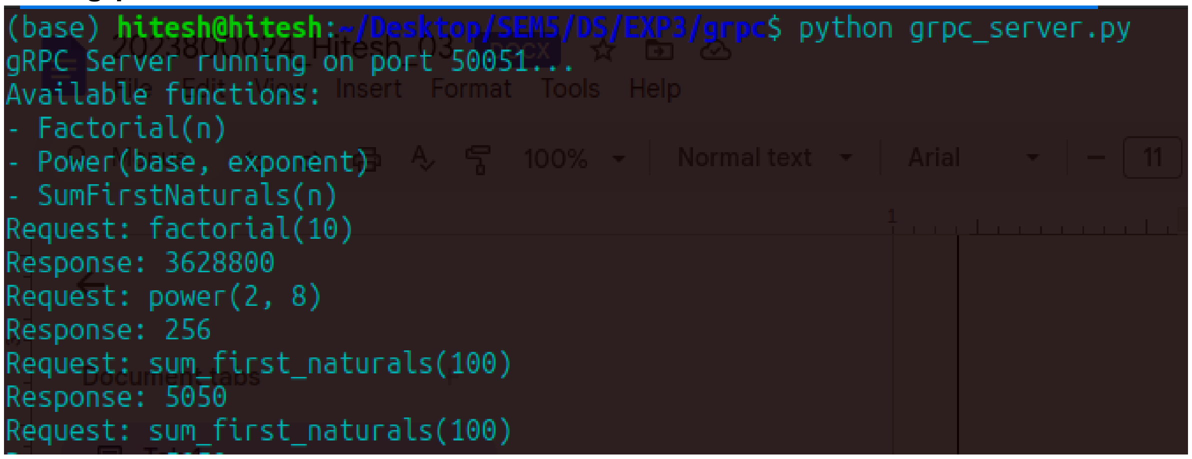


```
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/pyro5$ python pyro_server.py
Pyro5 Server is running...
Object URI: PYRO:obj_7586d4fac7434b0c84af952c815e8110@localhost:34697
Available functions:
- factorial(n)
- power(x, y)
- sum_first_n_naturals(n)
Press Ctrl+C to stop the server...
Request: factorial(10)
Response: 3628800
Request: power(2, 8)
Response: 256.0
Request: sum_first_n_naturals(100)
Response: 5050
Request: sum_first_n_naturals(50)
Response: 1275
Request: sum_first_n_naturals(50)
Response: 1275
```

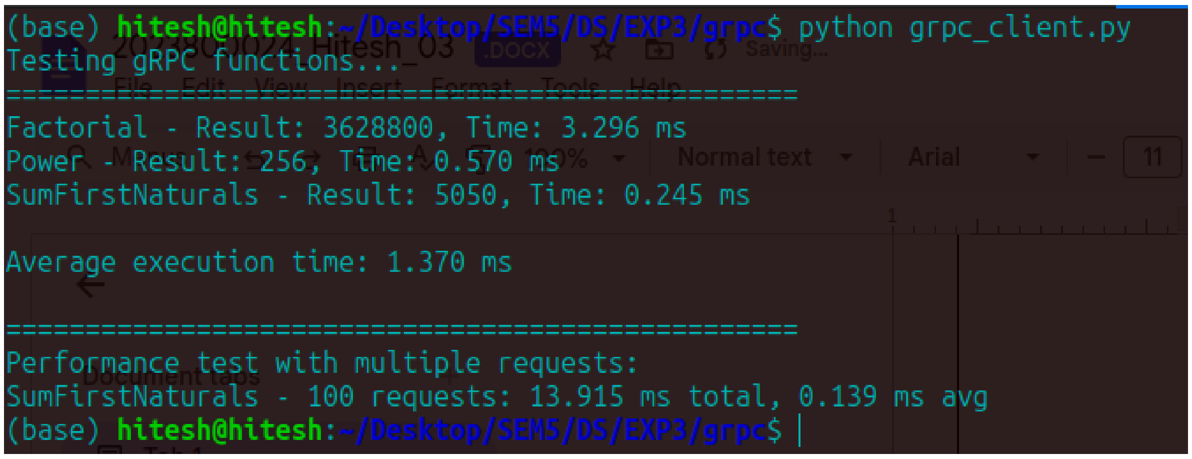


```
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/pyro5$ python pyro_client.py
Error: Could not find the name server. Make sure it's running with 'python -m Pyro5.nameserver'
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/pyro5$ python pyro_client.py
Found name server!
Found calculator service: PYRO:obj_7586d4fac7434b0c84af952c815e8110@localhost:34697
Testing Pyro5 functions...
==================================================
factorial(10,) = 3628800
Time taken: 1.052 ms
----------------------------------------
power(2, 8) = 256.0
Time taken: 0.156 ms
----------------------------------------
sum_first_n_naturals(100,) = 5050
Time taken: 0.273 ms
----------------------------------------
Average execution time: 0.494 ms

==================================================
Performance test with multiple requests:
sum_first_n_naturals - 100 requests: 5.495 ms total, 0.055 ms avg
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/pyro5$
```

PART grpc



```
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/grpc$ python grpc_server.py
gRPC Server running on port 50051...
Available functions:
- Factorial(n)
- Power(base, exponent)
- SumFirstNaturals(n)
Request: factorial(10)
Response: 3628800
Request: power(2, 8)
Response: 256
Request: sum_first_naturals(100)
Response: 5050
Request: sum_first_naturals(100)
```



```
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/grpc$ python grpc_client.py
Testing gRPC functions...
==================================================
Factorial - Result: 3628800, Time: 3.296 ms
Power - Result: 256, Time: 0.570 ms
SumFirstNaturals - Result: 5050, Time: 0.245 ms

Average execution time: 1.370 ms

==================================================
Performance test with multiple requests:
SumFirstNaturals - 100 requests: 13.915 ms total, 0.139 ms avg
(base) hitesh@hitesh:~/Desktop/SEM5/DS/EXP3/grpc$ |
```

Conclusion

This experiment has demonstrated a fundamental shift in how we approach inter-process communication (IPC) and distributed computing. By moving from the manual, low-level control of sockets to the high-level abstraction of Remote Procedure Call (RPC), we have successfully simplified the process of building distributed applications.

The core principle of RPC is to make a remote function call feel like a local one, effectively hiding the complexities of networking from the programmer. We achieved this by using different frameworks that automate the process of marshaling (packing) and unmarshaling (unpacking) data and handling network transport.

Our practical implementation with XML-RPC, gRPC, and Pyro5 showcased the diverse landscape of RPC solutions. While XML-RPC provides a simple, readable, and easy-to-understand entry point into the concept, its text-based format makes it less efficient. In contrast, gRPC represents a modern, high-performance solution that leverages a binary format and code generation, making it ideal for scalable, language-agnostic

	systems. Lastly, Pyro5 demonstrates a Python-specific, object-oriented approach that is highly intuitive for homogeneous environments. In conclusion, this experiment highlights the evolution of distributed systems, where we progressively abstract away complexities to build more efficient, reliable, and maintainable applications. The choice of the right RPC framework depends on the specific project requirements, balancing factors such as performance, language interoperability, and ease of use.
References	https://www.w3schools.com/xml/ Mam module