**EXECUTIVE SUMMARY**

**PURPOSE AND SCOPE:**

The purpose and scope of the project you described involve extending an existing system from Project #1 in the following two ways:

**Purpose:**

The primary purpose of this project is to improve the client-server architecture by enhancing communication efficiency through the use of RPC and enabling the server to handle concurrent requests.

**Scope**:

The scope of the project includes the following key components:

Integration of an RPC framework (e.g., Java RMI, Apache Thrift) to replace socket-based communication.

Modification of the server to be multi-threaded, enabling concurrent request handling.

Implementation of mutual exclusion mechanisms to ensure data consistency in a multi-threaded environment.

Support for concurrent PUT, GET, and DELETE operations from multiple client instances.

Testing to validate the correctness and performance of the enhanced system.

The expected outcome is an improved client-server system that can efficiently handle concurrent client requests using RPC while maintaining data integrity. The project's scope also covers the evaluation and validation of the system to ensure it meets the specified requirements and performs as expected.

**TECHNICAL OVERVIEW:**

**Java RMI (Remote Method Invocation):**

Java RMI is a Java API that enables communication between different Java Virtual Machines (JVMs) distributed across a network. It allows objects in one JVM to invoke methods on objects in another JVM, making it a powerful tool for building distributed systems.

Remote Interface: In Java RMI, you define a remote interface that extends the Remote interface. This interface declares the methods that can be invoked remotely. These methods should throw a RemoteException to handle network-related exceptions.

Remote Object: A remote object is a class that implements the remote interface. It extends UnicastRemoteObject to make it available for remote invocation.

RMI Registry: The RMI Registry is a simple naming service that binds objects to names. Clients can look up objects in the registry to obtain references for remote method invocation.

Stubs and Skeletons: Java RMI generates stubs (client-side) and skeletons (server-side) automatically. Stubs act as proxies for remote objects, and skeletons are responsible for dispatching remote method calls on the server.

Server Setup: On the server side, you create and publish remote objects in the RMI Registry. The server listens for incoming requests from clients and delegates them to the appropriate remote objects.

Client Setup: On the client side, you look up remote objects in the RMI Registry and call methods on them as if they were local objects. RMI handles the marshalling and unmarshalling of method arguments and return values.

RMI Workflow:

* The client looks up a remote object in the RMI Registry.
* The client calls methods on the stub (proxy) of the remote object.
* The stub serializes the method call and sends it to the server.
* The server's skeleton receives the method call, deserializes it, and invokes the method on the actual remote object.
* The server serializes the method result and sends it back to the client.
* The client's stub deserializes the result and returns it to the client code.

**Multi-threading:**

Multi-threading is a programming concept that allows a program to execute multiple threads (smaller units of a process) concurrently. Threads are lightweight, independent tasks that can perform operations simultaneously.

Thread: A thread is the smallest unit of execution in a program. Multiple threads can run concurrently within a single process.

Thread Safety: When multiple threads access shared resources (variables, data structures, etc.), special care must be taken to ensure that concurrent access does not lead to data corruption or inconsistent results. Synchronization mechanisms such as locks, semaphores, and monitors are used to achieve thread safety.

Concurrency vs. Parallelism: Concurrency is the ability of a system to handle multiple tasks at the same time, whereas parallelism is the simultaneous execution of multiple threads on multiple CPU cores.

Thread Lifecycle: Threads go through various states such as NEW, RUNNABLE, BLOCKED, WAITING, and TERMINATED. They are created, started, executed, and eventually terminated.

Thread Priorities: Threads can have priorities that affect the order in which they are scheduled by the operating system. Higher-priority threads are given preference.

Benefits of Multi-threading:

* Improved Performance: Multi-threading can lead to better performance, as multiple tasks can run concurrently, utilizing available CPU resources.
* Responsive User Interfaces: In applications with graphical user interfaces (GUIs), multi-threading can ensure that the interface remains responsive while background tasks are running.
* Efficient Resource Utilization: Multi-threading can make efficient use of resources, especially in servers and applications with high concurrency.
* Parallelism: In multi-core processors, multi-threading can achieve true parallelism, improving overall performance.

Challenges and Considerations:

* Thread Safety: Ensuring that shared resources are accessed safely by multiple threads is essential to avoid race conditions and data corruption.
* Synchronization Overhead: Synchronization mechanisms can introduce overhead and complexity in the code.
* Deadlocks: Poorly synchronized threads can lead to deadlocks where two or more threads are stuck and unable to make progress.
* Context Switching: Frequent context switching between threads can also introduce performance overhead.