

2024

Submitted to: Dr. Muhammad Imran

Net Centric Programming

Laiba Masood 04072013002

Nasir Hussain 04072013040

Report on Client-Server Application Using

TCP, UDP, and RMI

Contents

[Introduction 4](#_Toc156161431)

[TCP Implementation 4](#_Toc156161432)

[Implementation Details: 4](#_Toc156161433)

[Variations: 4](#_Toc156161434)

[Advantages/Disadvantages: 4](#_Toc156161435)

[UDP Implementation 5](#_Toc156161436)

[Implementation Details: 5](#_Toc156161437)

[Variations: 5](#_Toc156161438)

[Advantages/Disadvantages: 5](#_Toc156161439)

[RMI Implementation 6](#_Toc156161440)

[Implementation Details: 6](#_Toc156161441)

[Variations: 6](#_Toc156161442)

[Advantages/Disadvantages: 6](#_Toc156161443)

[General Observations 7](#_Toc156161444)

[Conclusion 7](#_Toc156161445)

## Introduction

The client-server application implements a simple interaction where the client sends input to the server, and the server responds based on the input. The code includes implementations using TCP, UDP, and RMI (Remote Method Invocation). Each approach has unique characteristics, and this report will discuss the implementation details, variations, and advantages/disadvantages of each.

## TCP Implementation

### Implementation Details:

* Server (TCP.Server):
* Listens on port 8888 for incoming TCP connections.
* Accepts each client connection and creates a new thread (`ClientHandler`) to handle client requests.
* The `ClientHandler` thread processes client input, performs the required actions, and sends the response back to the client.
* Client (TCP.Client):
* Takes user input from the console (either "time" or "testfile.txt").
* Establishes a TCP connection with the server.
* Sends user input to the server and displays the server's response.

### Variations:

* Multithreading with Thread Extension:

The TCP server utilizes multithreading to handle multiple clients concurrently. The ClientHandler class extends the Thread class, allowing for concurrent execution. Each client connection is processed in a separate thread, enabling efficient multitasking.

### Advantages/Disadvantages:

* Advantages:
* Reliable and connection-oriented.
* Well-suited for scenarios where ordered and reliable data transmission is crucial.
* Disadvantages:
* Higher overhead due to connection setup and maintenance.
* May not perform well in situations with high latency or low bandwidth.

## UDP Implementation

### Implementation Details:

* Server (UDP.UDPServer):
* Listens on port 8888 for incoming UDP packets.
* Creates a new thread (`UDPClientHandler`) for each client request.
* `UDPClientHandler` processes the client request and sends a UDP packet containing the server's response.
* Client (UDP.UDPClient):
* Takes user input from the console (either "time" or "testfile.txt").
* Sends a UDP packet with the user input to the server.
* Receives and displays the server's response.

### Variations:

* Concurrent Execution with Thread Extension:

Like the TCP implementation, the UDP server employs multithreading for concurrent execution. The UDPClientHandler class extends the Thread class, allowing for parallel processing of client requests. Each client request is handled independently in a separate thread.

### Advantages/Disadvantages:

* Advantages:
* Lower overhead as there is no connection setup/teardown.
* Suitable for scenarios where real-time communication is more critical than reliability.
* Disadvantages:
* Unreliable data delivery due to lack of acknowledgment and retransmission.
* Not ideal for applications requiring ordered and reliable data transfer.

## RMI Implementation

### Implementation Details:

* Server (RMI.RMIServer):
* Implements the `MessageProcessor` interface.
* Binds the server object to the RMI registry, making it available for remote invocation.
* Client (RMI.RMIClient):
* Uses RMI to look up the server object.
* Calls the `processMessage` method on the server object, passing user input.
* MessageProcessor Interface:
* Defines the remote method (`processMessage`) that the server implements.

### Variations:

* Remote Method Invocation (RMI):

RMI allows for remote invocation of methods on objects located on a server. The MessageProcessorImpl class extends UnicastRemoteObject and implements the MessageProcessor interface. UnicastRemoteObject provides the infrastructure for remote communication, and MessageProcessor defines the remote methods.

* No Direct Use of implements Runnable:

RMI does not directly use the Runnable interface for threading, as its focus is on remote method invocation. Thread management in RMI is often handled internally by the RMI runtime, and explicit implementation of Runnable is not necessary for the remote object.

### Advantages/Disadvantages:

* Advantages:
* Simplifies distributed computing by abstracting remote method calls.
* Provides a more natural and object-oriented approach to distributed systems.
* Disadvantages:
* Complexity in setup and configuration.
* May not be as efficient as lower-level protocols like TCP or UDP in certain scenarios.

## General Observations

* Thread Extension vs. implements Runnable:
* The use of thread extension (extends Thread) simplifies the creation of threads in both TCP and UDP implementations.
* While implementing Runnable is an alternative approach, extending Thread is straightforward and directly represents a thread.
* Trade-offs in Multithreading:
* Multithreading in TCP and UDP introduces concurrency for handling multiple clients, but it comes with potential challenges such as synchronization and shared resource management.
* RMI, being more abstract, may handle concurrency internally, abstracting away some of the complexities associated with multithreading.

## Conclusion

Each approach (TCP, UDP, RMI) in the client-server application serves specific use cases and comes with its advantages and disadvantages. TCP offers reliability and ordered data transmission but with higher overhead. UDP is lightweight and suitable for real-time communication but sacrifices reliability. RMI provides a higher-level, object-oriented approach to remote invocation but may introduce complexity.

The choice of the appropriate approach depends on the specific requirements of the application, such as the need for reliability, real-time communication, or a more natural object-oriented design. It's crucial to carefully consider the trade-offs and select the approach that best aligns with the project's goals.