LINUX MULTI-THREADED CLIENT-SERVER USING SHARED MEMORY PROJECT REQUIREMENT DOCUMENT

Table of Contents

[**1.** **INTRODUCTION** 3](#_Toc168434598)

[a) Overview 3](#_Toc168434599)

[b) Background 3](#_Toc168434600)

[**2.** **OBJECTIVE** 3](#_Toc168434601)

[a) Main Objective 3](#_Toc168434602)

[b) Specific Objectives 3](#_Toc168434603)

[**3.** **REQUIREMENTS** 3](#_Toc168434604)

[a) Functional Requirements 3](#_Toc168434605)

[b) Non-functional Requirements 4](#_Toc168434606)

[**4.** **SYSTEM DESIGN** 4](#_Toc168434607)

[a) Architecture 4](#_Toc168434608)

[**5.** **IMPLEMENTATION** 4](#_Toc168434609)

[a) Standard Libraries 4](#_Toc168434610)

[**6.** **DEPLOYMENT** 5](#_Toc168434611)

# **INTRODUCTION**

## Overview

This project aims to implement a basic client-server communication system using sockets and shared memory in the C programming language. The system allows multiple clients to connect to a server over a network and receive random numbers generated by the server.

## Background

Client-server architecture is a common paradigm used in networked systems, where clients request services from servers over a network. Sockets provide a mechanism for communication between processes over a network, while shared memory allows multiple processes to share data efficiently.

# **OBJECTIVE**

## Main Objective

The primary objective of this project is to demonstrate the implementation of a simple client-server communication system using sockets and shared memory.

## Specific Objectives

Establish TCP/IP socket connections between clients and a server.

Implement shared memory for efficient data exchange between the server and clients.

Allow multiple clients to connect to the server concurrently.

Generate random numbers on the server and share them with connected clients.

# **REQUIREMENTS**

## Functional Requirements

1. ***Client-Server Communication***

* Utilizes TCP/IP sockets for communication between the client and server.
* Allows multiple clients to connect to the server concurrently.
* Supports bidirectional communication, with the server sending random numbers to clients.

1. ***Shared Memory***

* Uses shared memory for efficient data sharing between the server and clients.
* Enables fast access to shared data without the need for inter-process communication overhead.

1. ***Multi-Threading***

* Implements multi-threading on the server to handle multiple client connections simultaneously.
* Ensures efficient resource utilization and responsiveness to client requests.

## Non-functional Requirements

1. ***Efficiency:*** The system should efficiently handle multiple client connections and share data without significant overhead.
2. ***Reliability:*** The system should be robust and capable of handling errors gracefully.
3. ***Scalability:*** The system should support a scalable number of client connections without degradation in performance.

# **SYSTEM DESIGN**

## Architecture

***1.Server Initialization***

* Creates a TCP/IP socket and binds it to a specified port for listening.
* Sets up a shared memory segment for storing random numbers.
* Initializes a semaphore for synchronization between processes accessing the shared memory.

***2.Client Connection Process***

* Establishes a connection to the server using TCP/IP sockets.
* Retrieves random numbers from the shared memory segment.

***3.Shared Memory Usage***

* Server generates random numbers and stores them in the shared memory.
* Clients access the shared memory to retrieve random numbers.

***4.Thread Management***

* The server employs multi-threading to handle multiple client connections concurrently.
* Each client connection is managed by a separate thread, ensuring parallel processing.

# **IMPLEMENTATION**

## Standard Libraries

* Include all the necessary library files.

## **DEPLOYMENT**

1. **Installation & Compilation**
   * Compile the server and client programs using a C compiler (e.g., gcc).
   * Ensure that the necessary libraries and header files are available.
2. **Execution**

* Run the compiled server program on the desired port.
* Run multiple instances of the client program to connect to the server.

1. **Usage**
2. ***Server***

* Start the server program, specifying the port number for communication.
* The server will begin listening for incoming client connections.

1. ***Client***

* Run the client program and provide the server's IP address and port number.
* The client will connect to the server and start receiving random numbers.

1. **Code Overview**
2. **Server-Side Components**

* generate\_random\_numbers(): Generates random numbers and stores them in shared memory.
* handle\_clients(): Handles multiple client connections concurrently.

1. **Client-Side Components**

* Connection setup with the server.
* Accessing shared memory to retrieve random numbers.

1. **Bash Script**

* A script to run multiple client instances concurrently.