**LINUX MULTI-THREADED CLIENTSERVER USING SHARED MEMORY**

**Table of Contents**

[Overview 3](#_Toc167976550)

[Objectives 3](#_Toc167976552)

[Declaration and Includes](#_Toc167976554) 3

[Code Functionality .......5](#_Toc167976555)

[Debbuging 6](#_Toc167976556)

[Conclusion](#_Toc167976557) 7

# **Overview**

This documentation presents the implementation of a multi-threaded server-client application designed to facilitate communication between a server and multiple clients using shared memory and semaphores. The server is responsible for handling multiple client connections simultaneously, where it receives messages from the clients, stores them in shared memory, and echoes the messages back to the clients. The use of shared memory allows for efficient data sharing, while semaphores ensure proper synchronization and data integrity.

# **Objectives**

The primary objective of this application is to demonstrate the practical use of inter-process communication (IPC) mechanisms in a multi-threaded environment. Specifically, it showcases the implementation of shared memory for data storage and semaphores for synchronization. The server manages multiple clients, ensuring each client's data is correctly processed and echoed back, maintaining consistency and synchronization across all interactions.

# **Declaration and Includes**

## Global Variables:

sem\_t \*semaphore :- Declares semaphore variable

int server\_fd :- Server file descriptor

int client\_count :- Counter to count clients

## Includes:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include<sys/ipc.h>

#include<unistd.h>

#include <semaphore.h>

# **Code Functionality**

## **Server Application (server.c)**

# Functionality:

* The server application listens for incoming client connections.
* It creates a new thread for each client connection to handle client communication.
* Each thread receives messages from its client, writes them to shared memory, and echoes the messages back to the client.
* Synchronization is ensured using semaphores to manage access to shared memory.

# Components:

* client\_handler: Thread function responsible for handling individual client communication. It receives messages from the client, writes them to shared memory, and echoes the messages back.

# Usage:

* Compile the server code: “ gcc -o server server.c ”
* Run the server: “./server”

## **Client Application (client.c)**

# Functionality:

* The client application connects to the server and sends messages.
* It receives echoed messages from the server and displays them.

# Components:

* “Main”: The main function of the client application. It connects to the server, sends messages, and displays echoed messages from the server.

# Usage:

* Compile the client code: “gcc -o client client.c”.
* Run the client: “./client”

**Debugging**

* Use the command in terminal “ ps -ef | grep server ” to find the server process.
* Then use “ pstree -p <pid> ” to view the process tree for the server.

# **Conclusion**

This project illustrates the effective use of shared memory and semaphores to manage inter-process communication in a multi-threaded client-server architecture. By implementing a server that handles multiple client connections, receives, stores, and echoes messages, the application demonstrates the importance of synchronization and efficient resource sharing. This approach not only ensures data consistency and integrity but also provides a scalable solution for managing concurrent client interactions in a networked environment.