RPS
learn..evolve

WIPRO NGA Program – LSP Batch

Capstone Project Presentation – 04 May 2024

Linux Multi-Threaded Client-Server Using Shared Memory

Presented by – Anirban Mazumdar

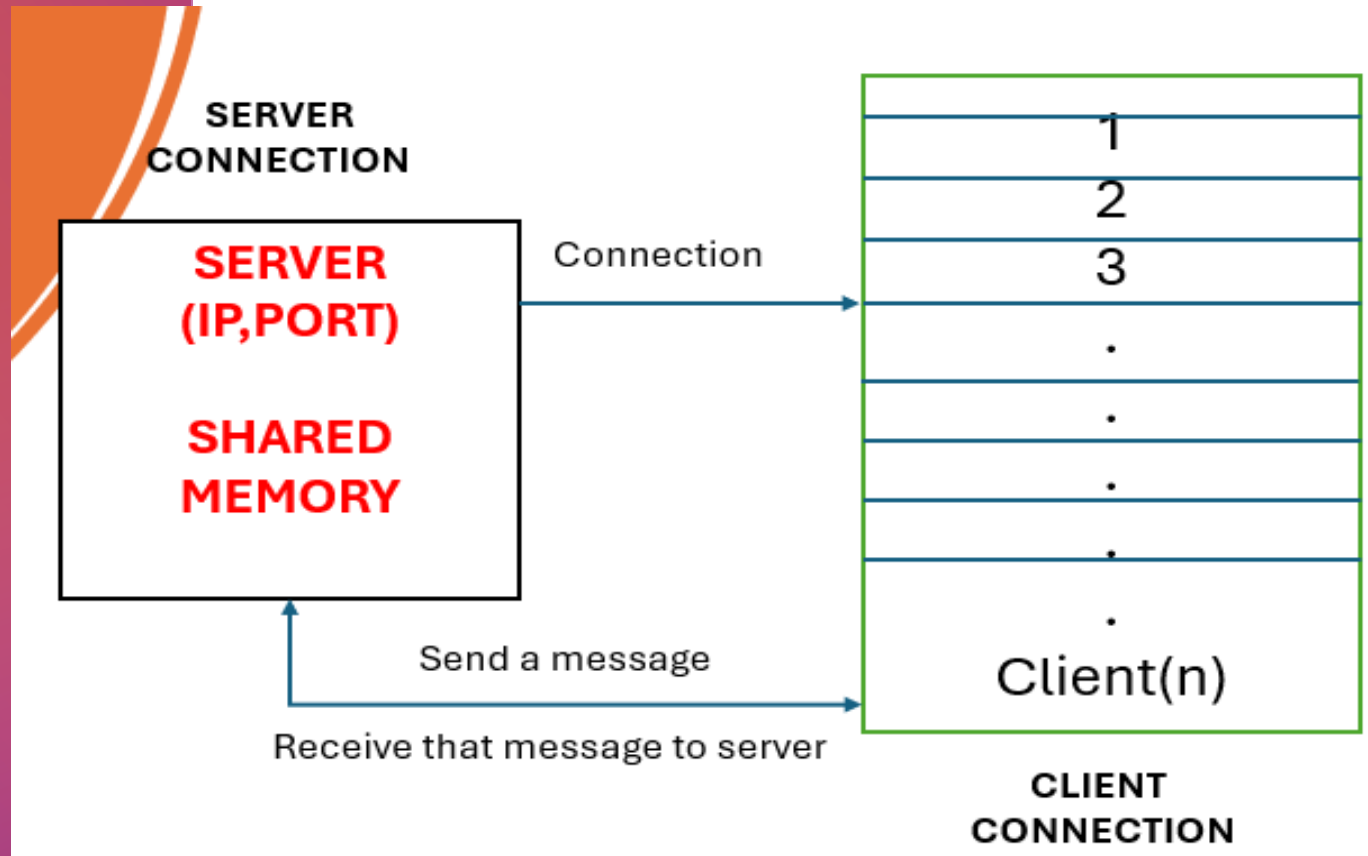
AGENDA

- Introduction
- Project Overview
- Code Functionality
- Test Cases
- Q&A

Introduction

- Client-server communication system using inter-process communication (IPC) mechanisms such as shared memory and semaphores.
- Server handles multiple clients simultaneously.
- Clients send and receives messages to and from the server. Server processes messages, updates shared memory, and responds to clients.
- Emphasizes IPC, shared memory management, and synchronization with semaphores

Design Overview



CODE FUNCTIONALITY: SERVER

Socket Setup:
- Bind socket to a specific port using bind().

Listening for Connections:
- Server listens for incoming connections using listen().

Client Handler Function:
- The client handler function manages communication with each client.
- Receive client messages, copy them to shared memory, and send back responses.

Handling Clients:
- Upon connection, spawn a new thread to handle each client request concurrently.

Shared Memory Communication:
- Initialize shared memory using for efficient data exchange between server and clients.

Connection Management:
- Detect client disconnection and close the socket upon termination.

Socket Bind

Shared Memory Setup

Create Thread(handleClient)=>ClientHandler)

Threads

For every n clients new thread is created (Client connection management)

Semaphore (wait and post)

Data Generator Thread – to send data to client via shared memory

Cleanup

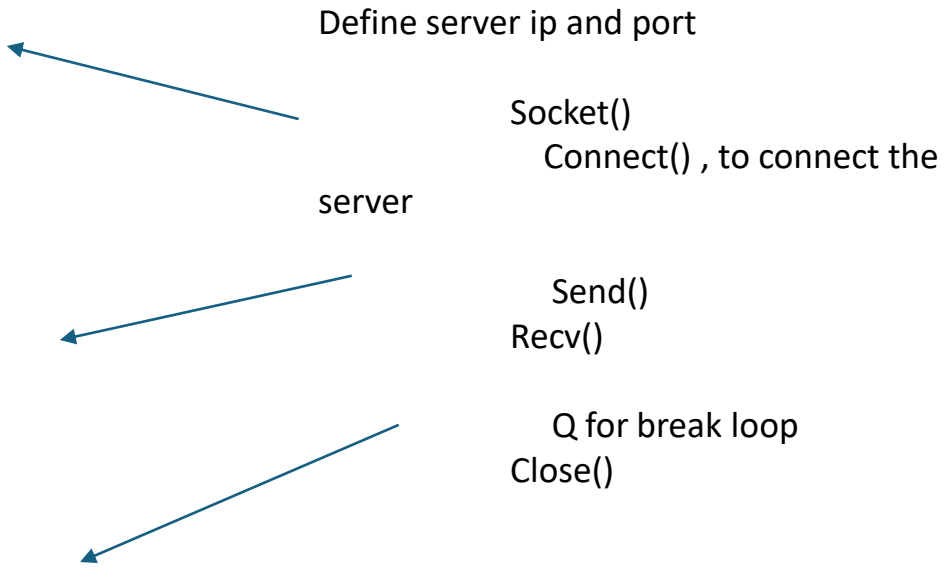
Release shared memory

Semaphores cleaned up

Closing Socket

CODE FUNCTIONALITY: CLIENT

- **Socket Setup and Connection:**
  - Create a socket and establish a connection to the server using connect().
- **Message Exchange:**
  - Send user-input messages to the server using send().
  - Upon receiving responses, print them using printf().
- **Connection Termination**
  - Quit the connection by typing 'Q' or 'q', closing the socket upon termination



Test Cases

1.Single Client Connection

- **Objective:** Verify server handles a single client connection.
- **Steps:** Start server, connect one client, send message, verify response.
- **Expected:** Server handles connection and message exchange correctly.

2.Multiple Client Connections

- **Objective:** Verify server handles multiple concurrent client connections.
- **Steps:** Start server, connect multiple clients, send messages concurrently.
- **Expected:** Server handles all connections and messages without data corruption.

3.Client Disconnection

- **Objective:** Verify server behavior on unexpected client disconnection.
- **Steps:** Start server, connect client, send message, disconnect client.
- **Expected:** Server detects disconnection and continues functioning correctly.

Test Cases

4.Shared Memory Synchronization

- **Objective:** Ensure proper synchronization of shared memory access.
- **Steps:** Start server, connect multiple clients, send simultaneous messages.
- **Expected:** Messages are written to shared memory without data corruption.

5.Maximum Client Connections

- **Objective:** Verify server's ability to handle maximum client connections.
- **Steps:** Start server, connect maximum clients, attempt extra connections.
- **Expected:** Server handles max clients correctly and refuses extra connections gracefully.


Questions

Thank you