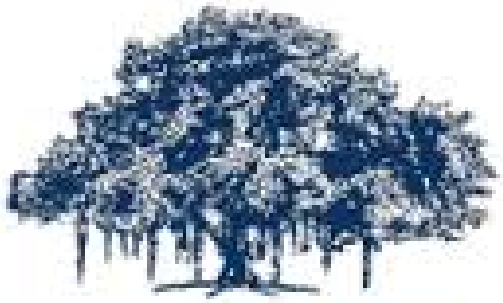
S25CS3.401 - Distributed Systems

Course Project

Distributed Shared Memory



INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

TEAM - 6

- **Manuj Garg - 2021101047**
- **Pranav Gupta - 2021101095**
- **Gowlapalli Rohit - 2021101113**

1. Introduction

The **Distributed Shared Memory (DSM) System** is a software-based approach that enables multiple clients to perform **READ** and **WRITE** operations on a shared memory space, distributed across multiple slave nodes. A master server coordinates these operations while ensuring **fault tolerance** and **data consistency**. This system is implemented using **Python** and **socket programming**, and is designed to operate on a single machine simulating a distributed environment.

2. Objectives

The primary goals of this project are:

- To develop a DSM system that allows multiple clients to interact with shared memory.
- To implement a **fault-tolerant** architecture using acknowledgment (ACK) bits from slave nodes.
- To ensure **consistency and efficiency** by distributing WRITE operations across a subset of slaves.
- To simulate the behavior of a distributed system on a single machine using socket programming.
- To incorporate a **backup master** mechanism to maintain system operations in case of a primary master failure.
- To conduct **scaled testing** to evaluate system performance under increased loads and multiple client/slave interactions.

3. System Architecture

The DSM system consists of three key components:

3.1 Client

- Connects to the master server to issue READ and WRITE commands.
- Displays responses received from the master server.
- Acts as an interface for users to interact with the distributed memory system.

3.2 Master

- Coordinates all operations between clients and slave nodes.
- Manages connections, tracks active slaves, and maintains data consistency.
- Distributes WRITE operations across a subset of slaves.
- Retrieves the latest values for READ operations and handles failures of slave nodes.
- Maintains a **backup master** to take over in case of primary master failure.

3.3 Slave

- Registers with the master and waits for incoming requests.
- Stores and retrieves data as instructed by the master.
- Sends ACK bits to confirm successful WRITE operations.

4. Functional Details

4.1 Master Process

- **Connection Management:** Listens for connections from clients and slaves.
- **WRITE Operations:**
    - Uses a **hash function** to distribute data across a subset of slave nodes.
    - Waits for ACK bits to confirm successful writes.
    - If a slave fails to respond, it is removed from the active list.
- **READ Operations:**
    - Retrieves data from the most recent successful WRITE operation.
    - If a slave fails to respond, the master retries until a valid response is received.
    - If data is not found in any slave, a "NOT_FOUND" message is returned.
- **Backup Master Handling:**
    - Upon failure of the primary master, clients and slaves connect to the backup master.
    - The backup master processes WRITE operations as usual.
    - For READ operations, if the requested key is missing in its records, it queries all slaves and updates its key-to-slave mapping based on majority response.

4.2 Slave Process

- Registers with the master upon startup.
- **WRITE Operations:** Stores received data and acknowledges the master.
- **READ Operations:** Responds with stored data when queried by the master.

4.3 Client Process

- Connects to the master server via a socket.
- Sends READ/WRITE requests and processes responses.
- Provides an interface for user interaction.

5. Fault Tolerance & Data Consistency

- **Fault Tolerance:**
    - ACK bits ensure reliable communication between master and slaves.
    - Non-responsive slaves are detected and removed from active operations.

    - Backup master mechanism ensures continued system functionality in case of primary master failure.
- **Data Consistency:**
    - The master tracks which slaves store which keys, ensuring READ operations always return the latest written value.
    - The backup master synchronizes missing key information by querying all available slaves.

6. Implementation Plan

- **Phase 1:** Develop socket communication between master, slaves, and clients.
- **Phase 2:** Implement basic READ/WRITE operations.
- **Phase 3:** Introduce fault tolerance mechanisms using ACK bits.
- **Phase 4:** Implement the backup master feature for reliability.
- **Phase 5:** Perform testing and optimization to ensure efficient data distribution.
- **Phase 6:** Conduct **scaled testing** to evaluate system performance under different workloads and varying numbers of clients and slaves.

7. Conclusion

This project demonstrates a distributed shared memory system with multiple clients and slaves, coordinated by a master server. With features such as fault tolerance, load-balanced data distribution, and backup master functionality, this DSM system ensures consistency and reliability. The project serves as a stepping stone for understanding distributed systems, data synchronization, and fault handling mechanisms in networked environments.